Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Bachelorarbeit 2008

Studiengang Wirtschaftsinformatik

Analyse komplexer Dokumente für die Extraktion und Indizierung von Texten und Bildern zur anschliessenden Informations-Suche

Student : Ivan Eggel

Dozent : Dr. Henning Müller

Vorwort

Diese Bachelor-Arbeit befasst sich mit der Analyse komplexer Dokumente für die Extraktion und Indizierung von Texten und Bildern, welche anschliessend für die Informationssuche verwendet werden.

Der Grund wieso dieses Thema von meinem Dozenten und Betreuer Dr. Henning Müller für eine Bachelor-Arbeit vorgeschlagen wurde, beruht auf einem konkreten Verlangen der WHO (World Health Organisation) für solch eine Lösung. Mit Hilfe des entwickelten Programmes soll ein Ansatz geschaffen werden, welcher erlaubt, nach Text und Bildern über ein Web-Interface suchen zu können. Vor allem im medizinischen Bereich in Entwicklungsländern kann dadurch das allgemeine Verständnis für eine sinnvolle Behandlung verbessert werden, indem z.B. eine textuelle oder visuelle Suche nach Frakturen vollzogen wird. Die aus der Suche hervorgegangenen Resultate könnten eventuell wertvolle Informationen über die Diagnose bis hin zur Behandlung liefern.

Ein anderer wichtiger Aspekt stellt die Wiederverwendbarkeit von Bildern und Texten dar. Die WHO besitzt bspw. Dokumente in mehreren Sprachen und in sehr unterschiedlichen Formaten (Word, PowerPoint, PDF, etc.). Häufig könnten dabei dieselben Bilder benutzt werden, aufgrund der öfters vorkommenden Unauffindbarkeit der Bilder müssen jedoch neue erstellt werden.

Die für die Indizierung und Suche unterstützten Dokumenttypen wurden aufgrund knapper zeitlicher Ressourcen nur auf die Formate Microsoft Office Word 97-2003, Microsoft Office Powerpoint 97-2003, Microsoft Office Excel 97-2003 und dem von der Firma Adobe Systems entwickelten Portable Document Format (PDF) gesetzt. Ich habe mich für diese Dateitypen entschieden, weil sie einerseits ein komplexes Format aufweisen (was die Extraktion von Inhalten erschwert), andererseits aufgrund der Tatsache, dass sie wohl den grössten Teil der im WWW (World Wide Web) verfügbaren Dateiarten darstellen. Diese Formate bilden auch eine konkrete Bitte der WHO.

Da diese Arbeit einen greifbaren Nutzen haben soll, hoffe ich, dass diese auch erweitert/verbessert wird und die Lösung damit praxistauglich macht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Code-Abschnitt-Verzeichnis

1 Einleitung

Dieses erste Kapitel beschäftigt sich primär mit allgemeinen Informationen, den vorgegebenen Zielen, sowie einer kurzen Zusammenfassung meiner Bachelor-Arbeit.

1.1 Motivation

Die Anzahl der in einer Unternehmung sich ansammelnden Dokumente in digitaler Form steigt in unserem Technologie-Zeitalter rasant an. Praktisch alle Dokumente werden heutzutage in elektronischer Weise verfasst und abgespeichert.

Je grösser diese Anzahl an Dokumenten ist, desto unübersichtlicher wird das ganze. Liegen auf einem Server tausende Dokumente, wird es sehr schwierig spezifisch im Dokument nach enthaltenen Wörtern oder gar Bildern zu suchen. Vor allem in Bezug auf die Bildsuche sind die vorhandenen Lösungen nur beschränkt hilfreich. Willkommen wäre eine Software, welche nach dem Inhalt der Bilder sucht (CBIR[1]) und nicht, wie normalerweise der Fall, nach Meta-Tags.

Diese Tatsache verlangt nach einer Applikation, welche alle Dokumente einer Unternehmung in einen Index aufnimmt und die Dokumente so für eine effektive Volltext- und Bilder-Suche aufbereitet. Diese Indizierung soll soweit wie möglich automatisiert ablaufen und nur geringe Kenntnisse des End-Users verlangen. Für die Suche ist ein Webinterface wünschenswert, welches jedem Mitarbeiter zur Verfügung steht und so einfach handzuhaben ist wie die Google-Suche.

1.2 Ziele

Viele Institutionen besitzen eine grosse Anzahl an Dokumenten, welche ein komplexes Format aufweisen (Word, Powerpoint, Excel, PDF, etc.). Um diese Informationen im Rahmen einer textuellen und visuellen Suche mittels existierender Tools wie Apache Lucene und dem GNU Image Finding Tool (GIFT) weiterverwenden zu können, ist es nötig, Text und Bilder der Dokumentquellen zu extrahieren und diese Teile jeweils separat zu indizieren.

[1] Ein Content Based Image Retrieval (CBIR) System führt die Bilder-Suche so aus, dass nur der eigentliche Inhalt des Bildes analysiert wird. Der Begriff Inhalt in diesem Kontext kann auf Farben, Formen Texturen oder andere Informationen bezogen sein, welche vom Bild selber abgeleitet werden können (Quelle: http://en.wikipedia.org/wiki/CBIR , Stand 24.10.2008).

- Bei dieser Bachelor-Arbeit handelt es sich in einem ersten Teil um eine automatisierte Extraktion von Text und Bilder verschiedener komplexer Formate der Dokument-Quellen.
- Diese Dokumente sollen anschliessend mittels existierender Tools (Apache Lucene für den Text, GIFT für die Bilder) indiziert werden.
- Zusammenarbeit mit der WHO welche ein konkretes Verlangen nach solch einer Lösung hat
- Eine maximale Automatisierung ist wünschenswert. Sowohl die Suche wie auch die Extraktion/Indizierung sollen über ein grafisches Web-Interface mit Ajax-Funktionalität ablaufen.

1.3 Zusammenfassung

Der praktische Teil meiner Bachelor-Arbeit befasst sich mit der Indizierung und Suche von Texten und Bildern komplexer Dateien.

Die im Rahmen dieser Arbeit entwickelte Lösung ist eine Web-Applikation. Aus diesem Grund ist es in einem ersten Schritt notwendig, die zu indizierenden Dokumente auf dem Server bereitzustellen. Für diese Bereitstellung habe ich folgende drei Verfahren implementiert:

- Upload des Dokuments durch den User
- Download eines im WWW öffentlich erreichbaren Dokuments mittels der Angabe des Pfades durch den User
- Einbeziehung eines Dokument-enthaltenden Verzeichnis auf dem Server mittels der Angabe des Server-Datei-System-Pfades durch den User

Nach der Bereitstellung eines Dokuments ist es in einem zweiten Schritt notwendig, die benötigen Inhalte aus dieser Datei zu extrahieren. Die relevanten Inhalte beziehen sich dabei auf den Text, die Bilder und den Autor des Dokuments.

Der nächste Arbeits-Schritt setzt sich als Ziel, die eben extrahierten Inhalte zu indizieren. Für die Indizierung Text-basierter Inhalte habe ich dabei die kostenlos vom Apache-Projekt zur Verfügung gestellte Open-Source Bibliothek Lucene verwendet. Mit Hilfe des von der Universität Genf entwickelten *GNU-Image-Finding Tool*s (GIFT), einem Open-Source CBIR-Programm für die Indizierung und Suche von Bildern, werden daraufhin alle im Dokument enthaltenen Bilder in einen Index aufgenommen.

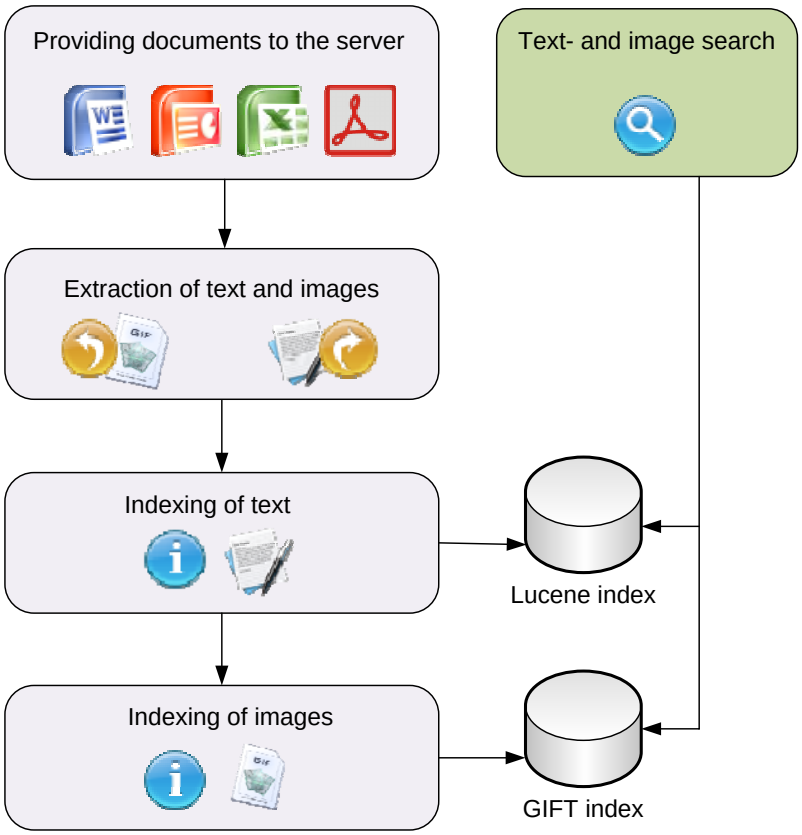


Abbildung 1-1 Aufbau meiner Applikation

Sobald die Text-und Bild-Inhalte des Dokuments für die Suche aufbereitet wurden, ist der End-User nun in der Lage, über das Web-Interface, eine sowohl textuelle (Inhalt und Autor) wie auch visuelle Suche (Bilder) zu lancieren. Diese beiden erwähnten Such-Arten wurden dabei in meiner Implementierung miteinander auf die Weise verknüpft, sodass es jeweils möglich ist von einem über die textuelle Suche gefundenen Dokument auf direkte Weise zu Inhalt-basiert ähnlichen Bildern zu gelangen. Zudem ist meine Applikation fähig, gefundene Bilder rückwirkend wieder einem Dokument zuzuordnen.

2 Gewählte Technologien

In diesem Kapitel werden die für meine Arbeit gewählten Technologien kurz vorgestellt und es wird erwähnt, wieso ich diese gewählt habe.

2.1 Programmiersprache Java

Abbildung 2-1 Java-Logo[2]

Da sich der weitaus grösste Teil meiner Bachelor-Arbeit mit der Entwicklung einer Applikation beschäftigt, war ich gezwungen, eine Programmiersprache zu wählen, mithilfe derer ich möglichst schnell und einfach ans Ziel gelange. Die Entscheidung fiel nach relativ kurzer Bedenkzeit auf die Programmiersprache Java. Es liegen mehrere Beweggründe vor, wieso ich mich für diese entschieden habe:

2.1.1 Gründe zur Wahl von Java

2.1.1.1 Bibliotheken

Unter dem in der heutigen Zeit immer wichtiger werdenden Aspekt des Rapid Application Development (RAD[3]), birgt das Vorhandensein zahlreicher Programmier-Bibliotheken eine hohe Zweckdienlichkeit. Nur schon die standardmässig in Java integrierten APIs[4] sind extrem nützlich und kaum mehr wegzudenken. Zudem gibt es eine Reihe an Drittanbietern zusätzlicher Bibliotheken (z.B. Apache Lucene), welche in ein Java-Projekt miteinbezogen

---

[2] Quelle: http://snapinfotech.com/yahoo_site_admin/assets/images/java_logo.15203252_std.png, Stand 02.10.2008

[3] Das Rapid Application Development sieht ein prototypisches Vorgehen vor, bei dem Anforderungen an eine Software gesammelt und möglichst schnell in ausführbaren Code umgesetzt werden. (Quelle: http://de.wikipedia.org/wiki/Rapid_Application_Development, Stand : 20.10.2008)

[4] Der Ausdruck API (Application Programming Interface) wird oftmals als Synonym für Bibliothek benutzt

werden können. Die Tatsache des Vorhandenseins solcher Code-Sammlungen beschleunigt die Dauer eines Projekts in höchstem Grade.

Ein Ziel meiner Arbeit stellt die Indizierung von Text mit Hilfe von Apache Lucene dar. Da Lucene zuerst nur als API für Java erhältlich war (mittlerweile gibt es auch Portierungen in andere Programmiersprachen), ist es die Version mit der höchsten Stabilität, den meisten Features und der grössten Community. Die Lucene-Bibliothek in Verbindung mit einer anderen Programmiersprache zu benutzen, wäre daher eher risikobehaftet.

2.1.1.2 Portabilität

Ein anderer entscheidender Faktor ist die Lauffähigkeit Javas auf fast sämtlichen Plattformen. Die Java Virtual Machine[5] (JVM) wurde auf sehr viele Betriebssysteme portiert, und ist daher sehr unabhängig. Sun versprach seit dem ersten Release: „Write Once, Run Anywhere[6]“. Dieser Slogan soll verdeutlichen, dass in Java geschriebener Code nur einmal geschrieben und kompiliert werden muss und trotzdem auf allen unterstützten Plattformen direkt lauffähig ist. Dieses Versprechen traf/trifft im Grossen und Ganzen auch tatsächlich zu.

2.1.1.3 Open Source[7] [8]

Ein zusätzlicher Grund zur Wahl der Programmiersprache Java besteht in ihrem Wesen selbst: Java ist Open-Source. Am 13. November 2006 veröffentlichte Sun seinen Compiler unter der GNU General Public License[9]. Am darauffolgenden September fand der Release des Codes der JVM sowie fast sämtlicher Java Klassen-Bibliotheken statt. Somit wurde Java grösstenteils Open-Source.

Einerseits stellt dies einen hohen Support und Stabilität/Optimierung sicher, andererseits gibt es aufgrund dieser Tatsache viele Programmierer welche Open-Source Bibliotheken für Java schreiben und die Sprache dadurch noch populärer machen

2.1.1.4 Kostenlos

Um auf einer Maschine Java ausführen zu können, bedarf es einer JVM. Diese steht kostenlos zum Download verfügbar und ist mittlerweile auf fast sämtlichen

---

[5] Eine Liste aller JVMs finden sie unter folgender Adresse: http://en.wikipedia.org/wiki/List_of_Java_virtual_machines.

[6] Quelle: http://de.sun.com/aboutsun/sun/, Stand 28.10.2008

[7] Open source (engl.) bzw. quelloffen ist eine Palette von Lizenzen für Software, deren Quellcode öffentlich zugänglich ist und durch die Lizenz Weiterentwicklungen fördert (Quelle: http://de.wikipedia.org/wiki/Open_Source, Stand 04.10.2008).

[8] Quelle: http://www.sun.com/2006-1113/feature/

[9] Die General Public License (oft abgekürzt GPL) ist eine von der Free Software Foundation herausgegebene Lizenz mit Copyleft für die Lizenzierung freier Software (Quelle: http://de.wikipedia.org/wiki/GNU_General_Public_License, Stand 20.10.2008).

PCs rund um den Erdball installiert. Das Vorhandensein zahlreicher kostenloser Java-Applikationsserver und Java-IDEs[10] figuriert als zusätzlicher Pluspunkt.

2.1.1.5 Persönliche Präferenz

Einer der wichtigsten Beweggründe zur Wahl von Java ist meine persönliche Vorliebe für diese Programmiersprache. Java besticht meiner Meinung nach durch die Einfachheit und die hervorragende Dokumentation der Java-APIs. Überdies stellte Java die Haupt-Programmiersprache während des Studiums dar.

2.2 Webtechnologien

Aufgrund des Vorteils einer Client-seitigen Plattform- und Standort-Unabhängigkeit war es eine Anforderung meiner Arbeit, die automatisierte Extraktion und Indizierung der Daten, sowie die damit verbundene Suche, über ein Webinterface verfügbar zu machen.

Da die Wahl der Programmiersprache bereits auf Java gefallen ist, war es nicht schwer, mich für eine Web-Technologie zu entscheiden. Die Wahl fiel auf JavaServer Faces (JSF[11]) in der Version 1.2.

2.2.1 Gründe zur Wahl von JSF

2.2.1.1 Java Web-Technologie

JSF wurde für Java entwickelt. Es war daher nicht nötig, eine völlig andere und mit Java-inkompatible Web-Technologie zu verwenden. Hätte ich eine zu Java unvereinbare Technologie (z.B. PHP[12]) gewählt, wäre es nötig gewesen, sämtliche relevanten Funktionen als WebService[13] bereitzustellen. Dies wäre unter dem Aspekt der benötigten Zeit sowie aus Performance-technischen Gründen praktisch untragbar gewesen.

---

[10] Eine integrierte Entwicklungsumgebung (Abkürzung IDE, von engl. *integrated development environment*, auch *integrated design environment*) ist ein Anwendungsprogramm zur Entwicklung von Applikationen (Quelle: http://de.wikipedia.org/wiki/Integrierte_Entwicklungsumgebung).

[11] Die JavaServer Faces Technologie erleichtert die Entwicklung von User-Interfaces für JavaServer-Applikationen (Quelle http://java.sun.com/javaee/javaserverfaces/, Stand 18.10.2008)

[12] PHP ist eine Skriptsprache mit einer an C angelehnten Syntax, die hauptsächlich zur Erstellung von dynamischen Webseiten oder Webanwendungen verwendet wird (Quelle: http://de.wikipedia.org/wiki/PHP, Stand 18.10.2008)

[13] Ein Webservice unterstützt die direkte Interaktion mit anderen Software-Agenten unter Verwendung XML-basierter Nachrichten durch den Austausch über internetbasierte Protokolle (Quelle: http://de.wikipedia.org/wiki/Webservice, Stand 20.10.2008)

2.2.1.2 MVC-Prinzip

JSF bietet den Vorteil einer strikten Implementierung des MVC[14]-Prinzips, welches eine klare Trennung von Geschäftslogik und Darstellung zulässt. Dadurch kommt es zu deutlich lesbarerem Code sowie einer erleichterten Erweiterbarkeit. Java Code wird somit niemals mit HTML- oder JSF-Tags direkt vermischt.

2.2.1.3 IDE-Unterstützung

Da die IDE NetBeans (Version 6.1) bereits zahlreiche Features (unter anderem einen grafischen Editor für den visuellen Aspekt) für JSF bereitstellt, wird die Arbeit natürlich nicht unerheblich beschleunigt. Überdies kann mit Hilfe der IDE der Java-Code direkt kompiliert und automatisch per Knopfdruck an den Server übermittelt werden.

2.2.2 Javascript/AJAX

Um meine Applikation ein wenig mit Web 2.0 Ambiance anzuhauchen, war es nötig, von der Technologie AJAX Gebrauch zu machen. AJAX ist die Abkürzung für *Asynchronous Javascript And XML*. Javascript wird clientseitig ausgeführt, dass heisst im Browser des End-Users. Mit Hilfe von AJAX ist es möglich mit JavaScript dem Server über eine URL[15] eine Anfrage zu schicken. Der Server seinerseits verarbeitet den Request und sendet Text zurück. Häufig wird als Antwort XML[16] zurückgeliefert, es ist jedoch kein Muss. Was der Client nun mit der Server-Response anstellt, bleibt ihm überlassen, häufig wird ein Teil der Webseite mit den neu-erhaltenen Daten aktualisiert. Dieser Ablauf geschieht völlig asynchron, das heisst, der End-User erhält Antwort vom Server muss jedoch nicht auf ein vollständiges Neu-Laden der Web-Seite warten.

---

[14] Das Model-View-Controller (MVC) Prinzip bezeichnet ein Architekturmuster zur Strukturierung von Software-Entwicklung in die drei Einheiten Datenmodell (engl. Model), Präsentation (engl. View) und Programmsteuerung (Quelle: http://de.wikipedia.org/wiki/Model_View_Controller, Stand 21.10.2008).

[15] Ein Uniform Resource Locator (URL) identifiziert eine Ressource über das verwendete Netzwerkprotokoll (beispielsweise http oder ftp) und den Ort (engl. location) der Ressource in Computernetzwerken (Quelle: http://de.wikipedia.org/wiki/Uniform_Resource_Locator, Stand 03.11.2008).

[16] Die Extensible Markup Language (engl. für „erweiterbare Auszeichnungssprache“), abgekürzt XML, ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdaten (Quelle: http://de.wikipedia.org/wiki/Extensible_Markup_Language), Stand 22.10.2008).

2.3 Applikations-Server Glassfish[17] [18]

Abbildung 2-2 Glassfish-Logo[19]

Für die Lauffähigkeit von JSF muss ein Applikationsserver installiert werden. Die Hauptaufgabe dieses Servers liegt darin, alle Client-Requests entgegenzunehmen und zu verarbeiten. Nach der Verarbeitung einer Anfrage sendet der Server den generierten HTML-Code an den Client (Browser) zurück.

Ich habe den Beschluss gefasst, die JavaEE5[20] Referenzimplementierung von Sun, den Glassfish Server in der Version v2ur2 (auch bekannt unter dem Namen Sun Java System Application Server 9.1) zu verwenden.

2.3.1 Gründe zur Wahl von Glassfish

Es liegen mehrere Gründe vor, wieso die Wahl auf den Glassfish-Applikations-Server gefallen ist. Die Hauptgründe dafür sind folgende:

- Lauffähigkeit auf verschiedenen Betriebssystemen
- Einfach zu installieren
- Gute Unterstützung des Glassfish in der NetBeans IDE
- Referenz-Implementierung des JavaEE5 Standards
- Open-Source
- Kostenlos
- Gute Performance

---

[17] Homepage Glassfish: https://glassfish.dev.java.net/

[18] Download Glassfish v2ur2: http://www.sun.com/software/products/appsrvr/get_it.jsp

[19] Quelle: https://glassfish-theme.dev.java.net/logo.gif

[20] Die Java Platform, Enterprise Edition (Java EE) basiert auf dem Fundament der Java Platform, Standard Edition (Java SE) und ist der Industrie-Standard für die Implementierung von Service-orientierter Architektur (SOA) und modernen Web-Applikationen (Quelle: http://java.sun.com/javaee/, Stand 27.10.2008)

2.4 Bibliotheken/Tools

2.4.1 Indizierung/Suche von Text mit Lucene[21]

Abbildung 2-3 Lucene-Logo[22]

Ein wichtiger Teil meiner Arbeit verkörpert die textuelle Indizierung/Suche. Um diesem Ziel gerecht zu werden, bedarf es einer guten Programmier-Bibliothek, welche einem einen soliden Grundpfeiler für diese Durchführung bietet. Wie in der Themenbeschreibung von Dr. Henning Müller bereits vorgeschlagen wurde, wird für diese Aufgabe das Java-API Lucene verwendet. In meiner Applikation verwende ich die Version 2.3.2[23].

Lucene ist eine high-performance, skalierbare Information-Retrieval (IR)-Bibliothek. Mit Lucene ist es möglich, Indizierungs- und Such-Funktionen in eine Applikation zu integrieren. Das Lucene-API ist ein reifes, kostenloses, Open-Source-Projekt welches in Java implementiert ist; Es ist Mitglied der beliebten Apache-Jakarta Familie, die unter der liberal Apache Software License lizenziert wurde und ist aktuell die populärste kostenlose Java IR-Bibliothek.

Lucene bietet ein simples aber äusserst leistungsfähiges API, welches nur minimale Kenntnisse der Volltext-Indizierung und Suche verlangt. Es ist nur nötig, über eine handvoll der vorhandenen Klassen Bescheid zu wissen, um von Lucene Gebrauch zu machen.

Lucene kümmert sich nicht um die Datenquelle, das Format, oder die Sprache, solange die Information in Text konvertiert werden kann. Im Konkreten bedeutet das: Die Benutzung Lucenes erlaubt die Indizierung und Durchsuchung von Dateien wie Webseiten, simplen Textdateien, Word-Dokumenten, oder PDF-Files. Einfach gesagt unterstützt Lucene jedes Format, von welchem textuelle Informationen extrahiert werden können.

2.4.2 Verwendete Bibliotheken für die Extraktion

Um überhaupt in der Lage sein zu können, Text und Bilder eines Dokuments zu indizieren, bedarf es einer Extraktion dieser beiden Modalitäten.

Im Zuge einer kurzen Google Recherche nach Java Bibliotheken für die Extraktion von Informationen meiner gewählten Dateiformate (Word, Powerpoint, Excel, PDF) bin ich mehr oder weniger direkt auf die Seiten von Apache POI (API

---

[21] Quelle: [Gospodnetic, Hatcher (2005)] Seiten 7,8

[22] Quelle: http://lucene.apache.org/images/lucene_green_300.gif, Stand 04.10.2008

[23] Download: http://www.apache.org/dyn/closer.cgi/lucene/java/

für MS Office Dokumente) und PDFBOX (API für PDF Dokumente) gestossen. Beide APIs sind Open-Source Projekte und daher sehr willkommen für meine Lösung.

2.4.2.1 Apache POI[24] [25]

Abbildung 2-4 Apache POI-Logo[26]

Apache POI ist ein Java API welches sich mit dem Zugang zu Microsoft Office File Formaten beschäftigt. POI ist das Akronym für Poor *Obfuscation Implementation* welches sich als *ärmliche Verschleierungs-Implementation* ins Deutsche übersetzen lässt. Das POI Projekt besteht aus mehreren Teil-APIs für die Manipulation verschiedener File-Formate welche auf dem *Microsoft OLE2 Compound Document Format* beruhen.

Für die Extraktion von Informationen bezogen auf meine Bachelor-Arbeit benutzte ich folgende Teil-APIs des POI-Projektes:

- **HWPF**[27]: Manipulation für Word-Dokumente der Versionen 97-2003
- **HSLF**[28]: Manipulation für Powerpoint Präsentationen der Versionen 97-2003
- **HSSF**[29]: Manipulation für Excel Tabellen der Versionen 97-2003

[24] Quelle: http://poi.apache.org/, Stand 14.10.2008

[25] Download: http://www.apache.org/dyn/closer.cgi/poi/

[26] Quelle: http://poi.apache.org/resources/images/project-logo.jpg,

[27] Akronym für "Horrible Word Processor Format" (http://poi.apache.org/hwpf/index.html)

[28] Akronym für "Horrible Slide Layout Format" (http://poi.apache.org/hslf/index.html)

[29] Akronym für "Horrible SpreadSheet Format" (http://poi.apache.org/hssf/index.html)

2.4.2.2 PDFBox[30] [31]

Abbildung 2-5 PDFBox-Logo[32]

PDFBox ist eine Open-Source Java Bibliothek für den Umgang mit PDF Dokumenten. Dieses Projekt erlaubt das Anlegen und Manipulieren von PDF Dateien, sowie das Extrahieren von Inhalten desselbigen Dateiformats. PDFBox enthält auch verschiedene Kommandozeilen Tools, welche ich jedoch in meiner Arbeit nicht benutzt habe. Ich habe mich ausschliesslich auf die Extraktion von Text und Bildern konzentriert.

2.4.3 Indizierung/Suche von Bildern mit GIFT[33]

Abbildung 2-6 GIFT-Logo

Eine zusätzliche Anforderung meiner Bachelor-Arbeit ist es, im Rahmen einer Inhalt-basierten visuellen Suche von Bildern der indizierten Dokumente, das Tool GIFT in meine Applikation zu integrieren.

Das CBIR-System GIFT ist ein Open-Source Franework für die Indizierung und Suche von Bildern. Bei der Bildsuche fungiert GIFT als Server, wobei es von einem Client aus möglich ist, eine sogenannte Query By Example (QBE) durchzuführen. Dies stellt einem die Möglichkeit bereit, die Abfrageresultate mithilfe eines Relevanz-Feedbacks zu verbessern. Für die Verarbeitung der Abfragen verlässt sich das Programm vollkommen auf den Inhalt der Bilder, was einen von der Verpflichtung befreit, die Bilder mit Anmerkungen (Meta-Tags) zu versehen.

---

[30] Quelle: http://www.pdfbox.org/, Stand 14.10.1008

[31] Download: http://sourceforge.net/project/showfiles.php?group_id=78314

[32] Quelle: http://www.pdfbox.org/images/Logo.gif, Stand 04.10.2008

[33] Quelle: http://www.gnu.org/software/gift/, Stand 12.10.2008

Zuständig für die Client-Server Kommunikation ist ein Protokoll namens MRML (Multimedia Retrieval Markup Language) welches auf XML basiert. Es existieren diverse Clients[34] für GIFT.

Was meine Arbeit anbelangt, habe ich mich auf die Implementierung des GIFT-Clients SnakeCharmer[35] von Zoran Pecenovic gestützt, dies aus dem einfachen Grund, weil dieser Client in Java geschrieben wurde. Obwohl die Implementierung schon etwas veraltet ist und nur als Java-Applet funktioniert, habe ich Teile daraus verwendet um eine Suche über ein JSF-Web-Interface zu implementieren. Die Aufgabe des Clients ist es, MRML-Messages zu verfassen und an den GIFT-Server zu senden. Der Server seinerseits schickt wiederum eine MRML-Nachricht an den Client zurück, welcher anschliessend die Nachricht parst und weiterverarbeitet.

2.4.4 IDE

Bei der Entwicklung von Applikationen wird es länger je wichtiger, eine gute IDE zu benutzen, vor allem aus dem Blickwinkel des RAD. Da ich schon seit Längerem mit Java vertraut bin, kann ich mit Sicherheit behaupten, dass die NetBeans IDE (Version 6.1[36]) meine Bedürfnisse im Rahmen dieser Bachelor-Arbeit am besten befriedigt.

Abbildung 2-7 NetBeans-Logo

Die NetBeans IDE ist ein Open-Source Entwicklungsumgebung welche die Entwicklung aller Java-Applikations-Typen unterstützt. Alle Funktionen der IDE werden in Modulen bereitgestellt.

Für meine Arbeit habe ich hauptsächlich folgende Module verwendet:

- Subversion
- Local-History
- IDE-Platform
- Visual JSF
- Glassfish
- Editing Files

[34] Eine vollständige Liste findet sich unter http://www.gnu.org/software/gift/mrml-clients.html.

[35] Download: http://viper.unige.ch/research/projects/GIFT/Charmer-0.2b.tgz

[36] Download: http://www.netbeans.org/downloads/

Um das Subversion-Modul verwenden zu können, musste zusätzlich ein Subversion-Client[37] der Firma Collabnet installiert werden. Gründe für die Wahl von NetBeans sind:

- IDE für Entwicklung jeglicher Art von Java Applikationen.
- Gute JSF Unterstützung mit grafischem Editor für das Page-Layout.
- Hohe Anzahl an zusätzlich downloadbaren Modulen.
- Hoher Funktionsumfang allgemein.
- Einfaches Deployment der Applikation durch die hervorragende Unterstützung des Glassfish-Servers.
- Integrierte Subversion-Unterstützung.
- Die wichtigsten Module sind bereits vorinstalliert.

[37] Dieses Programm steht unter folgender Adresse kostenlos zur Verfügung: http://www.collab.net/downloads/subversion/.

2.5 VmWare-Image

Abbildung 2-8 VMWare-Logo

Um meine Applikation vollständig portabel zu gestalten, habe ich den Entschluss gefasst, diese direkt auf einen virtualisierten Image für VMWare[38] zur Verfügung zu stellen.

2.5.1.1 Virtualisierung allgemein [39]

„Unter Virtualisierung versteht man eine abstrakte Ebene, die physische Hardware vom Betriebssystem entkoppelt und somit eine größere Auslastung der IT-Ressourcen und eine höhere Flexibilität ermöglicht.

Mit Virtualisierung ist es möglich, mehrere virtuelle Maschinen mit heterogenen Betriebssystemen einerseits isoliert, andererseits jedoch nebeneinander auf der gleichen physischen Maschine auszuführen. Jede virtuelle Maschine verfügt über einen eigenen virtuellen Hardware-Satz, wie z.B. RAM-Speicher, CPU, usw., auf den das Betriebssystem und die Anwendungen geladen werden. Das Betriebssystem erkennt, unabhängig von den tatsächlichen physischen Hardware-Komponenten, einen konsistenten und normalisierten Hardware-Satz.“

2.5.1.2 VmWare

Ein Anbieter für solche Virtualisierungs-Software ist die Firma VmWare. Durch das Installieren eines Betriebs-Systems auf ein VmWare-Image ist es möglich, dieses Image wie einen eigenständigen PC zu nutzen. Aufgrund der Tatsache, dass meine Applikation auf einem VmWare-Image zur Verfügung steht, ist es sehr einfach (ohne Installation), diese direkt auf einem beliebigen PC laufen zu lassen. Dazu muss lediglich eine Version des kostenlos zur Verfügung stehenden VmWare Players installiert sein.

2.5.1.3 Informationen zu meinem VMWare-Image

Installations-Optionen

Mein VmWare-Image wurde mit folgenden Optionen installiert:

---

[38] Homepage VmWare: http://www.vmware.com

[39] Quelle: http://www.vmware.com/de/virtualization/, Stand 03.11.2008

Option	Wert
Betriebs-System	Windows XP Professional SP2
Zugewiesener Arbeitsspeicher	1 GB
User-Name:	Administrator
Passwort	hevs07
Zusätzlich von mir installierte Software auf dem Betriebs-System	JDK 1.6[40] Glassfish-Server v2ur2
User-Name Glassfish	admin
Passwort Glassfish	hanswurst
Adresse der Web-Administrationskonsole des Glassfish	http://<IP-Adresse>:4848

Tabelle 2-1 Wichtige Informationen zum VmWare-Image

Starten der Applikation

Da der Glassfish-Server auf meinem VMWare-Image nicht als Service installiert wurde, muss dieser nach dem Aufstarten des Betriebs-Systems manuell gestartet werden. Dazu klicken Sie im Windows-Startmenü auf:

Programs > Sun Microsystems > Application Server 9.1 > Start Default Server

Nach diesem Schritt ist die Applikation lauffähig. Die Applikation kann danach unter folgender Adresse erreicht werden:

http:<IP-Adresse>:8080/TB_IVAN

Standort des Lucene-Index

Der Default Standort des Lucene-Index befindet sich im Verzeichnis *C:\Temp\testindex*. Dies ist wichtig zu wissen, falls einmal das darin enthaltene *write.lock*-File gelöscht werden muss. Mehr Informationen zum Zweck dieser Datei finden Sie unter Kapitel 5.2.3.

Standort der kompilierten Applikation und der Web-Verzeichnisse

Der Standort der Applikation und der Web-Verzeichnisse befindet sich im Verzeichnis *C:\Sun\AppServer\domains\domain1\applications\j2ee-modules\TEST4*. Die sich im Ordner *ApplicationProperties* befindende Konfigurations-Datei *App_Properties.properties* ist zuständig für grundlegende Einstellungen meiner Applikation. Hier wird bspw. der Standort des GIFT-Servers eingetragen. Ein Beispiel dieses Konfigurations-File finden sie im Anhang. Sollten Änderungen in dieser Datei vorgenommen werden, muss der Glassfish-Applikations-Server neu gestartet werden. Dazu klicken Sie im Windows-Starmenü auf:

---

[40] Download: https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/ViewProductDetail-Start?ProductRef=jdk-6u10-oth-JPR@CDS-CDS_Developer

Programs > Sun Microsystems > Application Server 9.1 > Stop Default Server

Dies verursacht die Terminierung des Glassfish-Servers. Für einen erneuten Start klicken Sie im Windows-Startmenü auf:

Programs > Sun Microsystems > Application Server 9.1 > Start Default Server

2.5.1.4 Öffentliche Erreichbarkeit meiner Applikation

Da meine Applikation öffentlich erreichbar sein muss (u.a. für die Verfechtung), habe ich dem Informatik-Dienst der Schule eine Kopie meines VMWare-Images übergeben. Die Verantwortlichen haben darauf hin mein Image auf einem VMWare-Server bereitgestellt, welcher öffentlich erreichbar ist (demilitarisierte Zone[41] der Schule).

Die öffentliche Adresse meiner Web-Applikation lautet:

http://153.109.124.56:8080/TB_IVAN

[41] Die in der demilitarisierten Zone (DMZ) aufgestellten Systeme werden durch eine oder mehrere Firewalls gegen andere Netze (z.B. Internet, LAN) abgeschirmt. Durch diese Trennung kann der Zugriff auf öffentlich erreichbare Dienste (Bastion Hosts mit z.B. E-Mail, WWW o.ä.) gestattet und gleichzeitig das interne Netz (LAN) vor unberechtigten Zugriffen geschützt werden (Quelle: http://de.wikipedia.org/wiki/Demilitarized_Zone, Stand 02.10.2008).

3 Bereitstellung der Dateien für die Extraktion / Indizierung

Bevor Dokumente überhaupt extrahiert und indiziert werden können, müssen diese natürlich verfügbar sein. Im Geltungs-Bereich meiner Applikation habe ich beschlossen, drei Arten für die Bereitstellung von Dokumenten zu unterstützen, welche alle über ein grafisches Web-Interface erfolgen sollen. Ziel dieser Bereitstellung ist es, Dokumente für die anschliessende Extraktion auf dem Applikations-Server in einem Verzeichnis abzuspeichern. Dieses Verzeichnis muss dabei zwingend öffentlich erreichbar sein, um den Usern die Möglichkeit des Downloads der im Verzeichnis enthaltenen Dokumente zu gewähren.

Für den User stellen die Schritte der Bereitstellung, Extraktion und Indizierung nur einen einzigen erkennbaren Arbeitsschritt dar, in programmier-technischer Hinsicht sind diese jedoch strikt getrennt.

3.1 Unterstützte Dokument-Formate

Wie schon im Vorwort angesprochen, habe ich mich dazu entschlossen, vier Dokument-Typen für die Extraktion zu unterstützen. Meine Applikation soll Informationen (Text und Bilder) von folgenden Dokument-Typen in der Lage sein zu extrahieren.

- Microsoft Word 97-2003
- Microsoft PowerPoint 97-2003
- Microsoft Excel 97-2003
- Portable Document Format (PDF)

Da nur die oben aufgeführten Dokument-Typen relevant sind, soll sich damit auch die Bereitstellung dieser Dateien auf dem Server nur auf diese beschränken.

3.2 Ablauf der Bereitstellung

Der Ablauf der Bereitstellung von Dokumenten spielt sich bei allen angebotenen Arten der Bereitstellung nach einem sehr ähnlichen Muster ab. Folgende Schritte werden dabei immer durchgeführt.

- Sicherstellung dass es sich um einen relevanten Dokument-Typ handelt.
- Austausch spezieller Zeichen im Dateinamen.
- Sicherstellung eindeutiger Dateinamen.

- Speicherung des Dokuments.

3.2.1 Sicherstellung der Bereitstellung eines relevanten Dokument-Typs

Um zum Vorneherein sicherzustellen, dass der User nur die obengenannten Dokument-Typen bereitstellen kann, soll dies gleich zu Beginn auf eine primitive Weise überprüft werden. Hierzu wird die Datei-Endung jedes bereitzustellenden Dokuments genauer unter die Lupe genommen. Die erlaubten Datei-Endungen beschränken sich auf nachstehende:

- doc (Word)
- ppt (PowerPoint)
- xls (Excel)
- pdf

Um zu gewährleisten, dass nur Dokumente mit korrekten Datei-Endungen bereitgestellt werden, habe ich in der Klasse `FileEndingChecker` des Pakets `td.ivaneggel.validation` die statischen Methoden `validate` und `validateWithoutZip` implementiert. Da die File-Endung *zip* für eine bestimmte Art der Bereitstellung (Hochladen) auch erlaubt ist, gibt es daher 2 Methoden für die Validation (Code-Abschnitt 3-1).

```
private static String endings[] = new String[]{"doc", "xls" ,"ppt",
        "pdf", "zip"};


public static  boolean validate(String filepath)
{
    filepath = filepath.toLowerCase();//Ignore case
    for(int i=0;i<endings.length;i++)
    {
       if (filepath.endsWith(endings[i]))
       {
          return true;
       }
     }
         return false;
    }


public static boolean validateWithoutZip(String filepath)
{
    filepath = filepath.toLowerCase();//Ignore case
    for(int i=0;i<endings.length-1;i++)
    {
       if (filepath.endsWith(endings[i]))
       {
           return true;
       }
    }
    return false;
}
```

Code-Abschnitt 3-1 Testen der Zugehörigkeit zu einem relevanten Dokument-Typ

3.2.2 Austausch spezieller Zeichen im Datei-Namen

Da die Dateien später in ein öffentliches Verzeichnis des Applikations-Servers gespeichert werden, soll der Datei-Name der Dokumente möglichst keine speziellen Zeichen (nicht UTF-8 Zeichen) enthalten. Spezielle Zeichen würden den programmier-technischen Aufwand zum Download des korrekten Files deutlich erschweren.

Ich habe dazu (zugegebenermassen kein ausgeklügeltes System) auf einfache Art die Datei-Namen-Modifikation im Falle eines File-Namens mit speziellen Charakteren implementiert. Die Methode welche diese Aufgabe übernimmt heisst `replaceSpecialCharsForFileName` und befindet sich in der Klasse `UtilClass` des Pakets `td.ivaneggel.utils`.

```
public static String replaceSpecialCharsForFileName(String
     filename)
{
     filename = filename.replace(' ', '_').replaceAll("ü", "ue").
               replaceAll("ä", "ae").replaceAll("ö",
               "oe").replaceAll("ß", "ss");

     return filename;
}
```

Code-Abschnitt 3-2 Austausch von Sonderzeichen im Datei-Namen

Aus dem Code-Beispiel ist zu entnehmen, dass die Methode `replaceSpecialCharsForFileName` Sonderzeichen des Datei-Namens austauscht und den neuen Namen als Return-Wert zurückgibt.

Folgende Zeichen werden dabei ausgetauscht:

Ursprüngliches Zeichen	Neue(s) Zeichen
Leerzeichen	Underscore
ü	ue
ä	ae
ö	oe
ß	ss

Tabelle 3-1 Ersetzung von Sonderzeichen

Diese Art des Austausches spezieller Zeichen ist sehr einfach gehalten. Für einen professionellen Gebrauch meiner Applikation müsste diese Methode überdacht werden. Zudem beschränkt dich der Austausch der Sonderzeichen nur auf die deutsche Sprache.

3.2.3 Sicherstellung eindeutiger Namen der Dokumente

In der Klasse `td.ivaneggel.utils.UtilClass` befindet sich die Methode `transformToUniqueFilePath`. Bevor ein Dokument abgespeichert wird, stellt diese Methode sicher, dass das abzuspeichernde Dokument nicht den gleichen Datei-Namen erhält wie ein anderes Dokument, welches bereits im Verzeichnis der abgespeicherten Dokumente lokalisiert ist.

Die statische Methode `transformToUniqueFilePath` erhält sowohl den Datei-Namen wie auch der Pfad des Verzeichnisses, in welchem die Datei später abgespeichert werden soll, als Parameter. Die Methode überprüft nun, ob eine Datei mit dem gleichen Namen bereits im Verzeichnis vorhanden ist. Ist dies der Fall, so wird der Datei-Name des abzuspeichernden Dokuments modifiziert. Der Stamm des File-Namens bleibt dabei gleich, es wird jedoch eine Zahl umgeben von Klammern hinzugefügt. Diese Nummer repräsentiert das *n-te* Vorkommen des

gleichen Dokument-Namens im Verzeichnis. Soll bspw. ein Dokument mit dem Namen *Sample.doc* abgespeichert werden und ist bereits im Verzeichnis der Dokumente vorhanden, wird der Name auf *Sample(2).doc* abgeändert. Wäre *Sample(2).doc* auch schon präsent, würde die Datei automatisch mit *Sample(3).doc* benannt. Um Ihnen einen genaueren Einblick zu gewähren, ist im untenstehenden Code-Abschnitt 3-3 die Methode `transformToUniqueFilePath` abgebildet,

```
public static String transformToUniqueFilePath(String filename,
       String uploadedDocumentsDirectoryAbsolutePath)
{
       String uploadedDocumentFileAbsolutePath=
           uploadedDocumentsDirectoryAbsolutePath+File.
           separator+filename;
       String newFilename="";
       File f = new File(uploadedDocumentFileAbsolutePath);
       int counter=2;
       int i = filename.lastIndexOf(".");

       //File exists already???
       //Rename Document if File already exists (add counter)
       while (f.exists() && f.isFile())
       {
           if ( i!=-1)
           {
               newFilename = filename.substring(0,i)+
                    "("+(counter++)+")"+filename.substring(i);
           }
           else
           {
               newFilename = filename+"("+(counter++)+")";
           }

           uploadedDocumentFileAbsolutePath =
                uploadedDocumentsDirectoryAbsolutePath+
                "/"+newFilename;
           f = new File(uploadedDocumentFileAbsolutePath);
       }

       return uploadedDocumentFileAbsolutePath;
}
```

Code-Abschnitt 3-3 Transformation zu einem eindeutigen File-Namen

3.2.4 Speicherung eines Dokuments

Wurden die vorangegangenen Schritte alle erfolgreich durchgeführt, ist das Dokument nun bereit, gespeichert zu werden. Wie Sie evtl. bereits aus dem Code entnehmen konnten, werden dabei alle Dokumente in das gleiche Verzeichnis des Servers abgespeichert. Die Wahl des Verzeichnisses wird dabei in einer Datei namens *App_Properties.properties* im Ordner *ApplicationProperties* des Applikations-Servers hinterlegt (eine Beispiel-Datei findet sich im Anhang). Diese Datei enthält alle wichtigen Einstellungen für meine Applikation. Wird darin eine Änderung vorgenommen, sollte der Glassfish-Server neu gestartet werden.

Die Wahl des Verzeichnisses für die Dokumente ist durch den Eintrag `upload-location` spezifiziert (Code-Abschnitt 3-4).

```
upload-location = /UploadedDocuments
```

Code-Abschnitt 3-4 upload-location

Normalerweise findet sich kein Anlass, diesen Pfad zu ändern. Will man den Pfad trotzdem ändern, sollte man dies tun, bevor Dokumente indiziert werden. Zu beachten ist, dass das neue Verzeichnis unverzichtbar öffentlich (im Kontext des Applikations-Servers) erreichbar ist.

Die Klasse `AppilcationBean1` im Paket `test4` liest verschiedene Eigenschaften des Property-Files aus. Über die Methode `getRealPathUploadedDocumentsDirectory` lässt sich der absolute Pfad zum Verzeichnis bequem holen.

3.3 Arten der Bereitstellung

Für die Bereitstellung der relevanten Dokumente, habe ich 3 verschiedene Verfahren implementiert:

- Hochladen einer Datei
- Download einer im WWW erreichbaren Datei auf den Server
- Einbeziehen eines ganzen Verzeichnisses auf dem Server

Diese drei Arten der Bereitstellung sollen im Folgenden nun genauer erläutert werden.

3.3.1 Hochladen einer Datei

Gemäss meiner Implementierung ist der End-User im Stande, im Web-Interface eine Datei hochzuladen. Dem User ist es dabei erlaubt einen relevanten Dokument-Typ oder eine *Zip*-Datei zu selektieren. Der Grund für die Unterstützung des Typs *Zip* ist die Möglichkeit einer Verpackung mehrerer relevanter Dateien innerhalb einer Datei, was das Hochladen und anschliessende Indizieren mehrerer Dateien gleichzeitig zulässt.

Wie es auf Web-Seiten üblich ist, wird der File-Upload über die Komponente *input file* des HTML-Standards realisiert. Diese Komponente ermöglicht einen Zugriff auf die lokale Festplatte, sodass der User sein Datei-System nach einer bestimmten Datei durchsuchen und anschliessend hochladen kann.

3.3.1.1 Hochladen eines relevanten Dokument-Typs

Zuständig für die den eigentlichen Upload des Files ist die Methode `upload` der Klasse `DocumentProvidingController` im Paket

`td.ivaneggel.jsfmanagedbeans`. Im Zuge der Betätigung des Upload-Buttons durch den User, wird der *Input-Stream* des upzuloadenden Files geholt und mittels eines *Output-Stream* auf die Hard-Disk des Servers geschrieben. Sobald sich die Datei persistent auf dem Server befindet, kann diese extrahiert und indiziert werden.

3.3.1.2 Hochladen einer Zip-Datei

Wählt der User eine *Zip*-Datei, wird dies über die Methode `isZip` der Klasse `FileEndingChecker` festgestellt. Die Aufgabe der Klasse `td.ivaneggel.utils.ZipExtractor` ist es nun, alle relevanten Dokumente (falls vorhanden) aus der *Zip*-Datei zu entpacken und gleichzeitig abzuspeichern. Relevante Dokumente werden dabei intern anhand der Methode `validateWithoutZip` gefunden. Korrespondiert eine Datei nicht mit einem unterstützter Dokument-Typ, wird sie ignoriert. Als Return-Value liefert die statische Methode `extract` des `ZipExtractors` den Pfad aller relevanten (abgespeicherten) Dokumente. Anhand dieser Pfade lässt sich später die Indizierung der Dokumente vornehmen.

```
if (FileEndingChecker.isZip(uploadedDocumentFileAbsolutePath))
{
      List<String> filePathsToIndex = null;
      filePathsToIndex = ZipExtractor.extract(new File
            (uploadedDocumentFileAbsolutePath),
             uploadedDocumentsDirectoryAbsolutePath);
}
```

Code-Abschnitt 3-5 Test der Datei-Endung mit Zip

3.3.2 Herunterladen eines öffentlich erreichbaren Dokuments auf den Server

Ist eine Dokument-Datei öffentlich im WWW verfügbar, erspart diese Lösung dem User das Herunterladen des Files auf die lokale Festplatte und das anschliessende Wieder-Hochladen desselben Files auf den Server. Der User tippt dabei die URL einer zu indizierenden Ressource in ein Text-Feld ein. Nach Betätigung des Submit-Buttons erfolgt der Download der Datei auf den Server.

Der Download einer Datei wird durch die Methode `download` der Klasse `DocumentProvidingController` gesteuert. Um den Download zu starten, wird die Methode mit den Argumenten Ziel-Datei (Ziel-Speicherort auf dem Server) und Quell-URL *(z.B. http://www.sample.hesso/Sample.doc)* versehen. Intern wird nun eine Verbindung zur Quell-URL der Datei hergestellt. Der *Input-Stream* dieser Verbindung ermöglicht es, die Datei zu lesen. Gleich während des Lesens des Files, wird es mit Hilfe eines *Output-Streams* auf die Server-Festplatte ins öffentliche Verzeichnis der Dokumente geschrieben.

```
public boolean download(File destination, String urlpath)
{

     BufferedInputStream bis = null;
     BufferedOutputStream bos = null;

     URL url = new URL(urlpath);
     URLConnection urlc = url.openConnection();

     bis = new BufferedInputStream( urlc.getInputStream());
     bos = new BufferedOutputStream( new FileOutputStream(
                    destination ) );
     int i;
     while ((i = bis.read()) != -1)
     {
          bos.write( i );
     }
     return true;
}
```

Code-Abschnitt 3-6 Methode für den Download einer Datei

3.3.3 Einbeziehen eines Verzeichnisses direkt auf dem Server

Eine nützliche Lösung für die gleichzeitige Indizierung mehrerer Dokumente ist das Einbeziehen eines Dokument-enthaltenden Verzeichnisses auf dem Server. Beherbergt der Server bspw. ein Verzeichnis, in welchem eine grosse Menge an relevanten Dokumenten enthalten ist, kann der User innerhalb des Web-Interfaces in einem Text-Feld den Pfad dieses Verzeichnisses (bezogen auf den Server) eingeben. Die einzige Aufgabe für die Bereitstellung der Dateien ist diesbezüglich die rekursive Durchsuchung des angegebenen Verzeichnis-Pfades nach unterstützten Dateitypen und das darauffolgende Kopieren dieser Dateien in den öffentlichen Dokument-Ordner.

```
public void setPathsOfRelevantFiles(File directory)
{
    File[] files = directory.listFiles();
    if (files==null) return;
    for (File file : files)
    {
        if (file.isDirectory())
        {
            setPathsOfRelevantFiles(file);
        }
        else
        {
            if (FileEndingChecker.validateWithoutZip
                    (file.getName())==true)
            {
                filepaths.add(file.getAbsolutePath());
            }
        }

    }
}
```

Code-Abschnitt 3-7 Methode zum finden aller relevanten Dokumente in einem Verzeichnis

Die Methode `setPathsOfRelevantFiles` der Klasse `IndexServerDirectory` im Paket `test4.Index` durchsucht das als Parameter erhaltene Verzeichnis rekursiv nach relevanten Dateien ab und speichert diese in eine Liste (Code-Abschnitt 3-7). Die Methode `copyToUploadLocation` im selben Paket kopiert nun alle zu berücksichtigenden Dokument-Dateien gemäss der vorhin zugewiesenen Liste in das öffentliche Dokument-Verzeichnis. Nicht-unterstützte Datei-Formate werden ignoriert.

4 Extraktion des Inhalts komplexer Dokumente

In diesem Kapitel wird beschrieben, wie die Extraktion des Inhalts (Text- und Bilder) vonstatten geht. Es wird aufgezeigt wie in etwa die Implementierung aussieht und welche Hilfsmittel verwendet wurden.

4.1 Komplexe Dokumentformate allgemein

Seit Anbeginn des Computerzeitalters werden Dateien verschiedenster Formate angelegt und abgespeichert. Anfänglich wurde höchstwahrscheinlich noch darauf geachtet, dass diese Dateiformate möglichst einfach zu lesen waren. Der Einzug der Kommerzialisierung hat, wie Sie sicherlich selber wissen, auch in der IT-Branche seine Spuren hinterlassen. Die damit verbundene Ausrichtung auf einen möglichst hohen Profit, hat diverse Firmen dazu veranlasst, komplexe Dateiformate zu entwickeln, welche nur von ihren eigenen Programmen gelesen und verarbeitet werden konnten. Will ein Programm einer anderen Firma dieses Dokumentformat lesen, müsste erheblich Zeit für die Analyse und Studie dieser Formate aufgebracht werden, da sie äusserst komplex aufgebaut sind und praktisch gesehen mit einer einfachen Verschlüsselung zu vergleichen sind. Glücklicherweise existieren eine handvoll Open-Source Projekte, welche zahlreiche Dateiformate (z.B. MS Word 97-2003) analysieren und für die Bearbeitung bzw. Extraktion von Inhalten ein Interface geschaffen haben, welches in zahlreichen Programmiersprachen als externe Bibliothek beigefügt werden kann. Das Problem hierbei ist jedoch einerseits, dass es längst nicht für jedes Dateiformat solch ein Tool gibt, andererseits dauert es immer eine gewisse Zeit (oft ziemlich lange) bis das Dokumentformat vollständig analysiert wurde.

In Zukunft wird der Trend wohl einen etwas anderen Weg einschlagen; weg von den komplexen Dateiformaten hin zu simplen Dateiformaten (allen voran XML[42]). Für XML beispielsweise gibt es wohl in allen erdenklichen Programmiersprachen mindestens einen Parser[43], mit dessen Hilfe sich relativ einfach Informationen eines Dokumentes auslesen lassen können.

---

[42] Die Extensible Markup Language (XML, ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdaten. (Quelle: http://de.wikipedia.org/wiki/Extensible_Markup_Language, Stand 08.11.2008)

[43] Computerprogramm, das in der Computertechnik für die Zerlegung und Umwandlung einer beliebigen Eingabe in ein für die Weiterverarbeitung brauchbares Format zuständig ist (Quelle: http://de.wikipedia.org/wiki/Parser, Stand 08.1..2008)

Nichtsdestotrotz werde ich mich im Rahmen meiner Bachelor-Arbeit nur mit der Informationsgewinnung *komplexer* Dateiformate befassen. Zum einen ist die Mehrheit der Dateiformate noch komplex, zum anderen besteht eine Herausforderung darin, dass es sicherlich schwieriger ist diese Informationen zu extrahieren, weil es in Zukunft höchstwahrscheinlich sowieso genügend Tools und Bibliotheken für Dateiformate geben wird, die auf XML oder sonstigen simplen Formaten basieren.

4.2 Konzept für die Extraktion

Um mein Projekt bezüglich der unterstützten Dateiformate erweiterbar gestalten zu können, musste ich mir ein möglichst einfaches Konzept ausdenken. Falls zu einem späteren Zeitpunkt einmal ein weiteres Dateiformat zur Extraktion von Inhalten hinzugefügt werden sollte, so ist es unumgänglich, dass nur eine minimale Veränderung am bestehenden Code vorgenommen werden muss und dass alles möglichst einfach und schnell vonstatten geht.

4.2.1 Grundlegender Aufbau

Ich habe mich dazu entschieden die abstrakte Klasse namens `ExtractedDocument` zu kreieren. Diese Klasse hat 3 abstrakte Methoden `getPictures`, `getText` und `getAuthor`, welche jeweils später von den Unterklassen implementiert werden. müssen

Es ist vorgesehen, dass jeder relevante Dateityp von `ExtractedDocument` erbt. Im Konkreten habe ich deshalb jeweils eine Klasse für die Word Dokumente (`WordDocument`), Powerpoint Pärsentationen (`PowerpointPresentation`), Excel Tabellen (`ExcelWorkbook`) und PDF Dokumente (`PDFDocument`) implementiert. All diese Subklassen erben von der abstrakten Superklasse `ExtractedDocument` (Abbildung 4-1 Klassenhierarchie für die Extraktion).

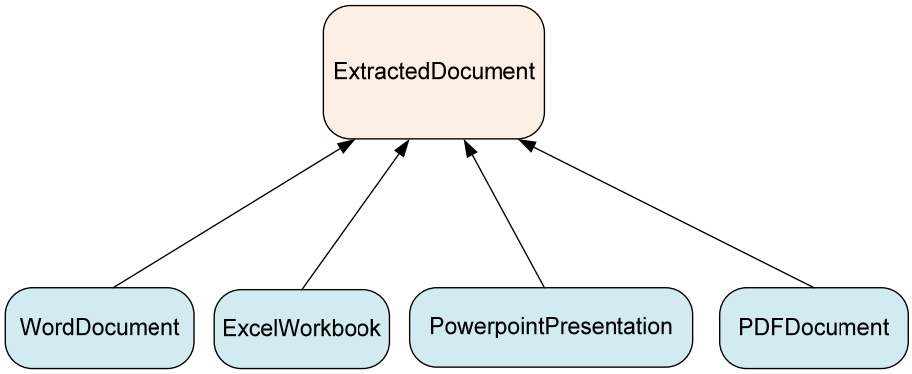


Abbildung 4-1 Klassenhierarchie für die Extraktion

Die abstrakte Klasse hat zudem eine implementierte Methode `saveImages`. Diese Methode ruft automatisch die Methode `getPictures` auf und extrahiert somit alle Bilder des jeweiliges Dokuments. Nach der Extraktion werden die Bilder in einen über das WWW öffentlich erreichbaren Ordner abgespeichert.

4.2.2 Instanziierung eines ExtractedDocument-Objekts

Ein `ExtractedDocument`-Objekt wird über die Klasse `FileEndingChecker` des Pakets `td.ivaneggel.validation` instanziiert. Die relevante statische Methode dieser Klasse heisst `getInstance` und sieht folgendermassen aus:

```
public static ExtractedDocument getInstance(String filepath)
{
    if (isWord(filepath))
    {
        return new WordDocument(filepath);
    }
    if (isExcel(filepath))
    {
        return new ExcelWorkbook(filepath);
    }
    if (isPowerpoint(filepath))
    {
        return new PowerPointPresentation(filepath);
    }
    if (isPDF(filepath))
    {
        return new PDFDocument(filepath);
    }
    return null;
}
```

Code-Abschnitt 4-1 Methode getInstance

Wie man aus obigem Code (Code-Abschnitt 4-1) entnehmen kann, verlangt die Methode einen String-Parameter, welcher den Pfad zum zu extrahierenden File repräsentiert. Durch den Pfad der Datei kann automatisch auf deren Endung (z.B. *doc*) geschlossen werden, wodurch seinerseits entschieden wird, um welchen Typ von Dokument es sich handelt. Angenommen der File-Pfad endet nun mit *doc*, wird automatisch die Klasse `WordDocument` instanziiert und gleich als Return-Wert zurückgeliefert. Somit wird eine einfache und sichere Art der Instanziierung sichergestellt. Im folgenden Code-Abschnitt ist ein Beispiel für eine solche Instanziierung gegeben.

```
ExtractedDocument doc =
    FileEndingChecker.getInstance(„C:\\Temp\\worddoc.doc“);
```

Code-Abschnitt 4-2 Instanziierung eines ExtractedDocument-Objekts

4.2.3 Extraktion des Textes

Über die Extraktion des Textes gibt es eigentlich nichts Spezielles zu erwähnen. Es wird ganz einfach der gesamte im Dokument enthaltene Text extrahiert. Zuständig ist hierfür die Methode `getText`. Wird die Methode `getText` von der

Superklasse `ExtractedDocument` aus aufgerufen wird in Polymorphie[44]-Manier automatisch die Methode für die gewünschte Klasse (z.B. `WordDocument`) aufgerufen. Der Text wird als einfacher String zurückgeliefert.

```
@Override
public String getText() throws TextNotExtractedException
{
	WordExtractor we=null;
	try
	{
		we = new WordExtractor(new
			FileInputStream(this.getFilepath()));

	} catch (Exception e )
	{
		e.printStackTrace();
		System.out.println(e.getMessage());
		throw new TextNotExtractedException(e);
	}

	return (we==null? "": we.getText());
}
```

Code-Abschnitt 4-3 Beispiel für die Implementation der getText-Methode

Das obenstehende Code-Abschnitt 4-3 Beispiel für die Implementation der getText-Methodeverdeutlicht die Extraktion von Text in Bezug auf ein Word-Dokument. Hierfür wurde die Methode `getText` der Superklasse `ExtractedDocument` mit Hilfe des APIs von Apache POI (siehe Kapitel 2.4.2.1) implementiert.

4.2.4 Extraktion der Bilder

4.2.4.1 Methode getPictures

Für die Extraktion der Bilder wird die Methode `getPictures` der abstrakten Superklasse `ExtractedDocument` implementiert. Das untenstehende Beispiel (Code-Abschnitt 4-4) veranschaulicht diese Methode in der Klasse `WordDocument`. Es werden systematisch alle Bilder des Dokuments extrahiert und direkt einer Liste des Typen `ExtractedPicture` hinzugefügt. Sobald alle Bilder in der Liste sind, wird diese als Return-Wert zurückgeliefert. Die Klasse `ExtractedPicture` ihrerseits ist nur ein Datenhalter, welcher den Namen des ursprünglichen Files (hier des Word-Files), die Dateiendung des Bildes und das Bild selbst in Byte-Form beherbergt.

[44] Polymorphie ist die Eigenschaft eines Bezeichners sich, in Abhängigkeit von der Umgebung in der er verwendet wird, unterschiedlich darzustellen. (Quelle: http://de.wikipedia.org/wiki/Polymorphie_(Programmierung), Stand 08.11.2008)

```
@Override
public List<ExtractedPicture> getPictures() throws
         ImageNotExtractedException
{
      List<ExtractedPicture> pictures = new
            ArrayList<ExtractedPicture>();
      try
      {
            HWPFDocument worddoc = new HWPFDocument(new
                  FileInputStream(this.getFilepath()));
            Iterator<Picture> i = worddoc.getPicturesTable().
                 getAllPictures().iterator();
            while(i.hasNext())
            {
                  Picture p = i.next();
                  String ending = p.suggestFileExtension();
                  if (ending==null) ending = "";
                  pictures.add(new

ExtractedPicture(p.getRawContent(),UtilClass.
                        getFilename(this.getFilepath()), ending));
            }
      }
      catch(Exception e)
      {
            e.printStackTrace();
            Syste.out.println(e.getMessage());
                  throw new ImageNotExtractedException(e);
      }
      return pictures;
}
```

Code-Abschnitt 4-4 Methode getPictures für die Extraktion der Bilder eines Dokuments

4.2.4.2 Methode saveImages

Da die Funktion `getPictures` alleine ziemlich nutzlos ist, wird sie innerhalb der Methode *saveImages* der Superklasse `ExtractedDocument` aufgerufen. Diese Methode vergibt die Filenamen für die Bilder und speichert diese ab. Zudem werden auch gleich Thumbnails mit-generiert und abgespeichert.

Mindestgrösse der Bilder und File-Endings

Die Methode `saveImages` iteriert die zuvor extrahierten Bilder und überprüft zuerst, ob das Bild eine ausreichende Grösse (in Pixeln) besitzt, da kleine Icons im Dokument oft irrelevant sind. Die Mindestgrösse der Bilder kann im Konfigurationsfile *App_Properties.properties* (Beispiel im Anhang) festgelegt werden. Falls ein Bild kleiner als die festgelegte Mindestgrösse ist, wird es ignoriert. Anschliessend wird überprüft, ob das Bild eine im Web anerkannte File-Endung besitzt (*jpeg*, *jpg*, *png*, *gif*). Falls dies nicht der Fall ist, wird das Bild wiederum ignoriert (siehe Code-Abschnitt 4-5).

```
if (UtilClass.isSupportedPictureEndingForWeb(ending)==false)
{
    continue;
}
```

Code-Abschnitt 4-5 Test ob Bild eine im Web anerkannte Datei-Endung besitzt

Namensgebung

Für die Namensgebung des Bildes wird (gemäss Code-Abschnitt 4-6) der Name des aktuellen Dokuments genommen. Zusätzlich dazu wird das Datum und die Anzahl Millisekunden seit 1970 angehängt. Um ganz sicher zu gehen (falls ein Schleifen-Durchlauf weniger als eine Millisekunde dauert, wird dem Bild noch eine ID mitgegeben. Diese ID repräsentiert die *n-te* Position des Bildes im Dokument. Somit ist es gänzlich ausgeschlossen dass 2 Bilder denselben Namen erhalten. Zudem kann exakt festgestellt werden, zu welchem Zeitpunkt das Bild gespeichert wurde.

```
pictureName = picture.getFilename()+"_"+UtilClass.
   getDateStringRepresentationForFileNames()+"_"+(id++)+"."+
   picture.getFileEnding();
```

Code-Abschnitt 4-6 Zusammensetzung eines Bilder-Namens

Speicherung

Sobald der Name vergeben wurde, wird das Bild in einem öffentlichen Verzeichnis abgespeichert. Der Pfad dieses Verzeichnisses ist ebenfalls Konfigurations-File hinterlegt.

Nachdem das jeweilige Bild abgespeichert ist, wird dem Attribut *pictureNames* (Liste) der Klasse `ExtractedDocument` der Name des Bildes beigefügt. Somit wird sichergestellt, dass nach der Extraktion und Speicherung der Bilder deren Dateinamen noch verfügbar sind, was später für die Indizierung wichtig sein wird.

Thumbnails

Der Enduser soll später bei einer Suche auch die Bilder jedes Dokuments sehen können. Um ihm einen Überblick darüber zu verschaffen, welche Bilder enthalten sind, sollen diese auf einer Seite als Thumbnails angezeigt werden. Hierfür musste ich natürlich auch Thumbnails generieren und abspeichern. Die File-Namen der Thumbnails bleiben gleich wie bei den Original-Bildern, sie werden jedoch in ein anderes Verzeichnis abgespeichert. Dieses Verzeichnis kann im Konfigurations-File festgelegt werden. Die Methoden zur Generierung von Thumbails befinden sich in der Klasse `Thumbnails`. im Paket `td.ivaneggel.utils`.

4.2.5 Extraktion des Autors

Auch die Extraktion des Autors gestaltet sich relativ simpel. Im folgenden Code-Snippet (Code-Abschnitt 4-7) wird die Extraktion des Autors in einem PowerPoint-Dokument veranschaulicht. Das benutzte API hierfür ist ebenfalls Apache POI (Kapitel 2.4.2.1).

```
@Override
public String getAuthor() throws
         TextNotExtractedException
{
      try
      {
            HSLFSlideShow powerpoint = new HSLFSlideShow
                  (this.getFilepath());
            return
            powerpoint.getSummaryInformation().getAuthor();
      }
      catch(Exception e)
      {
            throw new TextNotExtractedException(e);
      }

}
```

Code-Abschnitt 4-7 Methode getAuthor für die Extraktion des Autors eines Dokuments

4.3 Beispiel für die Extraktion aller relevanten Informationen

```
ExtractedDocument ed = FileEndingChecker.getInstance(path);
String text = ed.getText();
String author = ed.getAuthor();
ed.saveImages();
List<String> pictureNames = ed.getPictureNames();
```

Code-Abschnitt 4-8 Beispiel für die Extraktion aller relevanten Informationen

Das obige Code-Abschnitt 4-8 soll aufzeigen, wie nun alle für uns relevanten Informationen aus einem Dokument extrahiert werden:

- Instanziierung eines `ExtractedDocument`-Objekts über die Klasse `FileEndingChecker`. Als Argument wird der Methode der lokale Pfad des Files angegeben
- Zuweisen des Textes mit der Methode `getText` des soeben instanziierten Objektes
- Zuweisen des Autors mit der Methode `getAuthor`
- Extraktion und gleichzeitige Abspeicherung der Bilder, sowie Generierung und Abspeicherung der Thumbnails (`saveImages`)
- Zuweisen der eben abgespeicherten Bildernamen

Durch die Verschleierung der ganzen Komplexität mittels Vererbung braucht ein neu dazu-gestossener Programmierer kein Detail-Wissen, um Informationen aus den unterstützten Dokumentformaten zu extrahieren. Wie Sie oben im Code-Beispiel erkennen können, sind bloss 5 Kurze Schritte nötig

Nun sind wir am dem, Punkt angelangt, wo wir alle relevanten Informationen extrahiert haben. In einem späteren Schritt sollen diese dann in einen Index aufgenommen werden (Kapitel 5).

4.4 Erweiterung der unterstützten Dokument-Typen

Wie es allgemein bekannt ist, steckt vor allem die Technologie und damit logischerweise die IT-Branche in einem extrem dynamischen Prozess. Andauernd und in immer kürzer werdenden Zeitabständen werden neue Programme entwickelt und veröffentlicht. Dies führt auch unweigerlich dazu, dass sich immer wieder neue Dateiformate etablieren. Um diesem Wandel gerecht zu werden, habe ich mich für ein erweiterbares Interface meiner Informations-Extraktion entschieden.

Folgende Schritte sind nötig, um die Applikation mit einem neuen Dokument-Typ zu erweitern:

1. **Neue Klasse erstellen welche von der Klasse ExtractedDocument erbt**

```
public class NewFileFormat extends ExtractedDocument
```

Code-Abschnitt 4-9 Erstellen einer neuen Klasse

2. **Abstrakte Methoden implementieren**

```
public List<ExtractedImage> getImages(){...}
public String getText(){...}
public String getAuthor{...}
```

Code-Abschnitt 4-10 Implementierung aller abstrakten Methoden

Welches API für die Implementation verwendet wird ist absolut unbedeutend.

3. **In der Klasse FileEndigChecker die Klassenvariable endings[] aktualisieren**

```
private static String endings[] = new String[]{"doc",
"xls" ,"ppt", "pdf, “zip", “newFileEnding”};
```

Code-Abschnitt 4-11 String Array mit allen unterstützten Datei-Endungen

Wichtig ist, dass die die neue Datei-Endung als zweit-letztes Element im Array (vor *zip*) platziert wird.

4. **In derselben Klasse eine statische Methode is<Dokumenttyp> implementieren**

```
public static boolean isnewDocumentType(String
  filepath)
{
    return (filepath.endsWith(endings[n]));
}
```

Code-Abschnitt 4-12 Methode zum Testen der Datei-Endung

Die Methode kann kopiert werden (z.B. von `isZip`). Anschliessend wird der Name der Methode geändert und *n* auf die Position (Integer-Wert) des neuen Dokumenttyps im `endings`-Array festgelegt. In diesem Beispiel wäre *n=4*.

5. **Die existierende Methode getInstance (auch inder gleichen Klasse) aktualisieren**

```
if (isnewDocumentType(filepath))
{
    return new newFileFormat(filepath);
}
```

Code-Abschnitt 4-13 Methode für die Instanziierung

Der Methode `getInstance` wird eine neues *if-Statement* hinzugefügt, welcher die im Punkt 4 erstellte Methode aufruft. Als Instanz wird dann ein Objekt der Klasse, welche unter Punkt 1 erstellt wurde, zurückgeliefert.

Dies ist bereits alles, um die Im Rahmen dieser Bachelor-Arbeit entwickelte Lösung um einen weiteren Dateityp zu erweitern.

5 Indizierung mit Lucene

Sobald die Extraktion des Textes vollstreckt ist, ist es an der Zeit, den Text zu indizieren. Dieses Kapitel sieht daher vor, die Indizierung mit Lucene sowohl im Allgemeinen, wie auch bezogen auf meine Applikation etwas genauer zu betrachten. Die untenstehende Grafik (Abbildung 5-1) beschreibt, wie Lucene im Groben funktioniert.

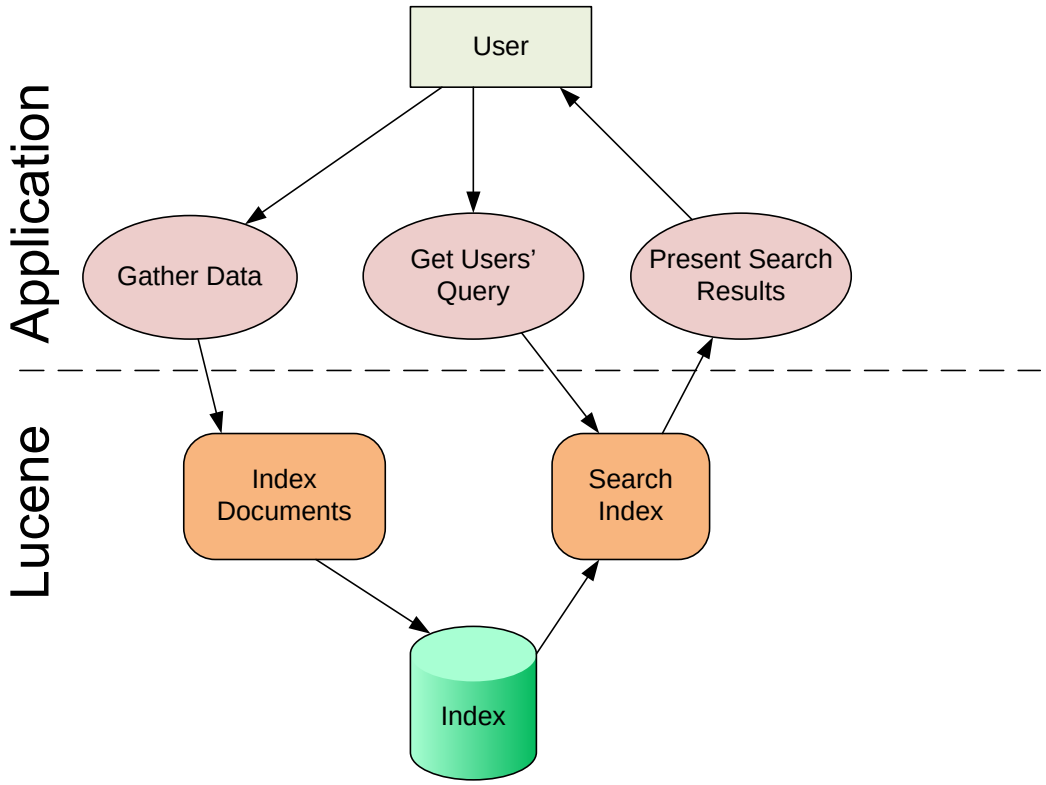


Abbildung 5-1 Funktion von Lucene

5.1 Allgemeine Informationen zur Indizierung[45]

Um Dateien zu finden, in welchen ein bestimmtes Wort oder ein bestimmter Satz enthalten ist, muss ein ausgeklügeltes Konzept vorhanden sein. Ein primitiver Ansatz wäre es, jedes File nach dem gegebenen Wort oder dem gegebenen Satz sequentiell zu durchsuchen. Diese Lösung hätte jedoch eine gewaltige Auswirkung auf die Performance, da das Lesen der gesamten Files sehr viel Zeit in Anspruch nähme. Um dies zu vermeiden, wird das Prinzip der Indizierung angewandt. Um grosse Mengen an Text effizient und schnell zu durchsuchen, ist es von grossem

[45] Quelle: [Gospodnetic, Hatcher (2005)], Seiten 10,11

Nutzen den Text in ein Format zu konvertieren, welches erlaubt ihn schnell zu durchsuchen, wobei gleichzeitig das langsame sequentielle Lesen eliminiert werden soll. Dieser Konvertierungs-Prozess wird Indizierung genannt. Der aus diesem Prozess erfolgte Output ist ein sogenannter Index.

Man stelle sich unter einem Index eine Datenstruktur vor, welche einen willkürlichen und schnellen Zugang zu den enthaltenen Wörtern zulässt. Das Konzept hinter dieser Struktur verhält sich analog zu einem Index am Ende eines Buches, mit dessen Hilfe sich schnell die Seiten zu einem bestimmten Thema lokalisieren lassen. Im Falle von Lucene ist ein Index eine speziell entworfene Datenstruktur, welche typischerweise auf dem Dateisystem als Satz von Dateien angelegt wird.

5.2 Indizierung mit Lucene allgemein

5.2.1 Hauptklassen für die Indizierung[46]

Für die Durchführung einer Indizierung mit Lucene werden vor allem folgende Klassen benutzt:

- `IndexWriter`
- `Directory`
- `Analyzer`
- `Document`
- `Field`

5.2.1.1 Klasse IndexWriter

Der `IndexWriter` ist die zentrale Komponente des Indizierungs-Prozesses. Diese Klasse erzeugt einen neuen Index und fügt Dokumente zu einem existierenden Index hinzu. Mit Hilfe des `IndexWriters` erlangt man ein Schreibrecht für den Index, es ist jedoch nicht möglich den Index damit zu lesen oder zu durchsuchen.

5.2.1.2 Klasse Directory

Die `Directory` Klasse repräsentiert den Standort des Lucene Indexes. Es ist eine abstrakte Klasse die es für deren Subklassen möglich macht, den Index in der Form abzuspeichern, wie es am besten passt (nach Wunsch des Programmierers).

Eine Directory-Instanz wird dem `IndexWriter` im Konstruktor mitgegeben, damit dieser weiss, welchen Index er nutzen soll. Lucene enthält bereits 2 konkrete Subklassen, welche von Directory erben. Zum einen ist das die Klasse `FSDirectory`, die den Index auf dem lokalen File-System abspeichert und zum

[46] Quelle: [Gospodnetic, Hatcher (2005)], Seiten 18, 19, 20, 21, 22

anderen die Klasse `RAMDirectory`, welche es ermöglicht, den Index im Arbeitsspeicher zu halten. Die letztere Implementation ist jedoch nur für kleinere Indizes geeignet, mit dem Anspruch, den Index löschen zu dürfen, sobald die Applikation beendet wird. Die Vorteile eines `RAMDirectory` sind die enorm hohe Indizierungs- und Such-Geschwindigkeit. Im Rahmen meiner Bachelor Arbeit verwende ich jedoch nur die Klasse `FSDirectory`. Somit ist der Index persistent und bleibt damit erhalten, auch wenn die Applikation gewollt oder weniger gewollt terminiert wird.

5.2.1.3 Klasse Analyzer

Bevor es zur Indizierung des Textes kommt, wird dieser an den sogenannten `Analyzer` weitergeleitet. Der `Analyzer`, welcher ebenfalls dem Konstruktor des `IndexWriters` übergeben wird, ist verantwortlich für die Extraktion bestimmter relevanter *Tokens* aus dem übergebenen Text, wobei der Rest des Textes ignoriert wird. Der `Analyzer` ist eine abstrakte Klasse, Lucene hat jedoch mehrere konkrete Klassen davon bereits implementiert. Beispielsweise gibt es Implementationen welche sogenannte Stopp-Wörter ignoriert (häufig gebrauchte, irrelevante Wörter wie z.B. a, *an*, *the*, *in*, etc.) oder alle Grossbuchstaben zu Kleinbuchstaben konvertiert, was eine eventuelle spätere Suche Case-insensitive gestalten soll. Der in meiner Arbeit benutze `Analyzer` ist die in Lucene bereits integrierte Klasse `StandardAnalyzer`, welche mehr oder weniger alle gebräuchlichen Fähigkeiten besitzt, den Text in sinnvolle Tokens aufzuteilen. Jedoch ist dieser `Analyzer` nur für die englische Sprache geeignet. Stopp-Wörter anderer Sprachen (z.B. deutsch) werden nicht ignoriert. Je nach Sprache sollte daher nach einem passenden `Analyzer`[47] gesucht werden, oder (falls nötig) selber einer implementiert werden, was jedoch mit einem nicht unerheblichen Zeitaufwand verbunden wäre.

5.2.1.4 Klasse Document

Die Klasse `Document` repräsentiert eine Kollektion sogenannter `Fields` (siehe nächster Abschnitt). `Fields` eines Dokuments stellen Daten dar welche das Dokument selber repräsentieren (z.B. den Text). Sie können jedoch auch Meta-Information wie z.B. den Autor beherbergen. Das ursprüngliche Dokument (z.B. ein Word-Dokument) ist für Lucene völlig irrelevant, da es sich nur mit Text (Klasse `java.lang.String`) befasst.

5.2.1.5 Klasse Field

Wie bereits erwähnt enthält ein `Document` eines Index eines oder mehrere `Fields`. Solch ein `Field` korrespondiert zu einem Datenteil, der abgefragt oder abgerufen werden kann. Indiziert man besipielsweise Dokumente mit dem Field-Namen *content* und *author*, kann in einer späteren Suche nach einem bestimmten

[47] Zusätzliche Analyzer findet man unter der Adresse http://lucene.apache.org/java/docs/developer-resources.html

Wort im Feld *content* gesucht werden. Lucene listet darauf hin alle Dokumente auf, welche das bestimmte Wort im Field *content* enhalten. Da das Suchergebnis in Form von Dokumenten zurückgeliefert wird, kann automatisch bei jedem gefundenen Dokument ebenfalls das Feld *author* ausgelesen werden. Dieses Verhalten ähnelt sehr demjenigen einer Datenbank.

Lucene bietet folgende Optionen für die Speicherung des Feldes an:

Option	Funktion
`Field.Store.YES`	Inhalt des Feldes wird gespeichert
`Field.Store.NO`	Inhalt des Feldes wird nicht gespeichert

Tabelle 5-1 Optionen für die Speicherung eines Feldes

Für die Indizierung eines Feldes sind folgende Optionen verfügbar.

Option	Funktion
`Field.Index.TOKENIZED`	Inhalt des Feldes wird indiziert und dem Analyzer übergeben.
`Field.Index.NO`	Inhalt des Feldes wird nicht indiziert. Nach dem Inhalt dieses Feldes kann später nicht gesucht werden.
`Field.Index.NO_NORMS`	Inhalt des Feldes wird indiziert, jedoch ohne Normen. Ohne Normen bedeutet, dass bei der Indizierung keine Lucene-spezifischen Normen für die Relevanz berechnet werden. Zudem wird das Feld nicht dem Analyzer übergeben und das Feld selber hat keine Längenbeschränkung.
`Field.Index.UN_TOKENIZED`	Inhalt des Feldes wird indiziert aber nicht dem Analyzer übergeben.

Tabelle 5-2 Optionen für die Indizierung eines Feldes

5.2.2 Kurzbeispiel für die Indizierung eines Dokuments

Das nachfolgende Beispiel soll kurz und prägnant vermitteln, wie die Indizierung mit Lucene im Groben funktioniert.

Der untenstehende Code (Code-Abschnitt 5-1 Instanziierung der Klasse Document) zeigt eine simple Instanziierung eines neuen Lucene-Dokuments.

```
Document doc = new Document();
```

Code-Abschnitt 5-1 Instanziierung der Klasse Document

Es wird ein neues Feld mit dem Namen *content* erzeugt. Als zweites Argument wird dem Konstruktor der Klasse `Field` der eigentliche Text übergeben. Mit `Field.Store.YES` wird dem Field mitgeteilt, dass sein Inhalt gespeichert werden soll. Das letzte Argument des `Field`-Konstruktors legt die Art der

Indizierung fest. In diesem Fall (Code-Abschnitt 5-2) wird also ein Feld angelegt, welches gespeichert und indiziert wird.

```
String myText = “Just a sample text”;
Field content = new Field(“content”, myText, Field.Store.YES,
     Field.Index.TOKENIZED);
```

Code-Abschnitt 5-2 Erzeugen eines Field-Objekts

Ähnlich geht die Indizierung des Autors vonstatten (Code-Abschnitt 5-3), mit der Ausnahme, dass das Feld nicht indiziert wird. Konkret heisst das: Nach dem Autor kann nur gesucht werden.

Führe ich bspw. eine Suche nach dem Wort *sample* im Feld *content* durch und erhalte das Ergebnisdokument, kann ich nicht an den Namen des Autors gelangen, weil er nicht abgespeichert wurde. Führe ich jedoch eine Suche nach dem Autor durch, so kann ich problemlos den dazugehörigen Text des Dokuments abfragen.

```
String theAuthor = “Ivan Eggel”;
Field author = new Field(“author”, theAuthor, Field.Store.YES,
        Field.Index.NO);
```

Code-Abschnitt 5-3 Erzeugen eines Field-Objekts

In diesem Schritt (Code-Abschnitt 5-4) werden der `Document`-Instanz beide vorher erzeugten Felder beigefügt.

```
doc.add(content);
doc.add(author);
```

Code-Abschnitt 5-4 Hinzufügen von Feldern zu einem Dokument

Eine `Directory`-Instanz wird über die statische Methode `getDirectory` der Klasse `FSDirectory` erzeugt (Code-Abschnitt 5-5). Wie bereits erwähnt bedeutet `FSDirectory`, dass der Index auf dem lokalen File-System abgespeichert ist/wird. Als Argument wird der Methode der Pfad des Index mitgegeben.

```
Directory d = FSDirectory.getDirectory(„C:\\temp\\testindex“);
```

Code-Abschnitt 5-5 Erzeugen eines Directory-Objekts

Im Code-Abschnitt 5-6 sehen Sie die Instanziierung eines `Analyzers`. Es wird der `StandardAnalyzer` verwendet, welcher bspw. die Stopp-Wörter ignoriert und alles in Kleinbuchstaben umwandelt. Beim Indizierungsprozess wird somit unser Beispieltext `myText` von *Just a sample text* auf den String *just sample text* reduziert.

```
Analyzer a = new StandardAnalyzer();
```

Code-Abschnitt 5-6 Instanziierung eines Analyzers

Anschliessend wird eine `IndexWriter`-Instanz erzeugt (Code-Abschnitt 5-7). Als Argumente erhält der Konstrukor, das `Directory`- und das `Analyzer`-Objekt. Zusätzlich verlangt der Konstruktor einen `Boolean`-Parameter, welcher angibt, ob der Index erzeugt werden soll, falls er noch nicht existiert. Darauffolgend wird

dem Index das vorhin erzeugte Dokument mit der Methode `add` hinzugefügt. Der Indizierungs-Prozess ist somit abgeschlossen

```
IndexWriter iw = new IndexWriter(d, true, a);
iw.addDocument(doc);
```

Code-Abschnitt 5-7 Instanziierung der Klasse IndexWriter

5.2.3 Indizierung/Persistierung mit Lucene bezogen auf meine Applikation

Im Paket `td.ivaneggel.indexing.lucene` befindet sich die Klasse `StandardFileIndexer`. Die Aufgabe dieser Klasse ist es, Informationen des extrahierten Dokuments entgegenzunehmen und zu indizieren.

Die Klasse `InformationToIndex` im selben Paket hat den Zweck, die eben extrahierten Informationen zwischenzuspeichern, um diese anschliessend dem Konstruktor der Klasse `StandardFileIndexer` zu übergeben. Folgende Angaben werden in einem `InformationToIndex`-Objekt gespeichert:

- Adresse des Dokuments
- Name des Dokuments
- Text
- Abstract
- Autor
- Liste mit Bildernamen

All diese Informationen können mit dem unter Kapitel 4 beschriebenen Code extrahiert werden. Sobald die Informationen eines Dokuments extrahiert wurden, wird ein `InformationToIndex`-Objekt mit den nötigen Daten über den Konstruktor erzeugt. Ist dies erfolgt, ist es nun an der Zeit den `StandardFileIndexer` zu instanziieren

5.2.3.1 Bereitstellung eines einzigen IndexWriters

Die in einer Web-Applikation typischerweise resultierenden konkurrierenden Threads haben auch einen Einfluss auf den `IndexWriter`. Es sollte unbedingt darauf geachtet werden, dass nur eine einzige Index-modifizierende Operation gleichzeitig ausgeführt werden darf. Wird dies nicht beachtet, so besteht ein sehr hohes Risiko an Inkonsistenz des Index. Daher wird dringend geraten, nur eine einzige `IndexWriter`-Instanz offen zu halten[48].

Aus diesem Grund habe ich den `IndexWriter` im Applikations-Kontext meiner Web-Applikation platziert. Der Applikations-Kontext wird durch die Klasse `test4.ApplicationBean1` repräsentiert. Sobald die Web-Applikation gestartet

[48] Quelle: [Gospodnetic, Hatcher (2005)], Seite59, 60

wird, wird diese Klasse automatisch instanziiert und steht somit der gesamten Applikation (jedem einzelnen Thread) zur Verfügung. Da der `IndexWriter` nun in diesem `ApplicationBean1` beherbergt ist, wird dieser nur ein einziges Mal instanziiert. So kann es kann zu keiner Inkonsistenz des Index kommen.

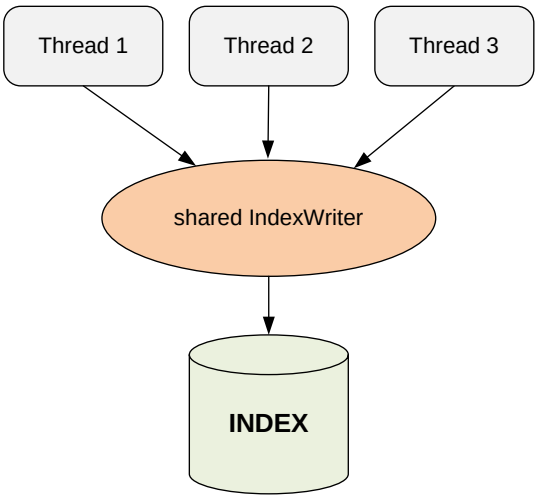


Abbildung 5-2 Sharing des IndexWriters unter mehreren Threads[49]

5.2.3.2 Prozess der Indizierung

Der Indizierungs-Prozess meiner Applikation spielt sich folgendermassen ab (Code-Abschnitt 5-8):

- Die `IndexWriter` Instanz wird über das `ApplicationBean1` geholt.
- Ein `InformationToIndex` Objekt wird anhand der extrahierten Daten des ursprünglichen Dokuments erzeugt.
- Für die Instanziierung des `StandardFileIndexers` ist es nötig, dem Konstruktor sowohl den IndexWriter-, wie auch das `InformationToIndex`-Objekt zu übergeben.
- Mit der Instanzmethode `indexInformation` werden die Informationen nun indiziert.

```
IndexWriter iw = getApplicationBean1().getIndexWriter();
InformationToIndex iti = new InformationToIndex(text, author,
   filename, etc.);
StandardFileIndexer stf = new StandardFileIndexer(indexWriter,
   iti);
stf.indexInformation();
```

Code-Abschnitt 5-8 Prozess der Indizierung

[49] Grafik wurde sinngemäss übernommen aus: [Gospodnetic, Hatcher (2005)], Seite 61

Verwendete Felder

Für die Indizierung/Persistierung eines Lucene-Dokuments, werden folgende Felder mit folgenden Eigenschaften benötigt:

Name des Fields	Store	Index
content	`Field.Store.YES`	`Field.Index.TOKENIZED`
fileaddress	`Field.Store.YES`	`Field.Index.NO`
filename	`Field.Store.YES`	`Field.Index.UN_TOKENIZED`
author	`Field.Store.YES`	`Field.Index.TOKENIZED`
abstracttext	`Field.Store.YES`	`Field.Index.NO`
imagename	`Field.Store.YES`	`Field.Index.UN_TOKENIZED`

Tabelle 5-3 im Ramen der Applikation verwendete Felder

Methode indexInformation

Wie bereits erwähnt, werden durch die Methode `indexInformation` der Klasse `StandardFileIndexer` die im `InformationToIndex`-Objektz enthaltenen Daten indiziert. Intern (in der Methode selbst) müssen diese zuerst abgerufen werden (siehe Code-Abschnitt 5-9).

```
String text = iti.getText();
String fileAddress = iti.getFileAddress();
String author = iti.getAuthor();
String abstractText = iti.getPictureNames();
List<String> imagenames = iti.getPictureNames();
```

Code-Abschnitt 5-9 Abfrage der Daten eines InformationToIndex-Objekts

Nach dieser Prozedur wird gemäss Code-Abschnitt 5-10 ein Lucene-Dokument mit den entsprechenden Feldern erstellt.

```
Document doc = new Document();
if (text!=null && !text.equals(„“)
{
     Field content = new Field(“content”, text,
             Field.Store.YES,
             Field.Index.TOKENIZED);
     doc.add(content);
}
```

Code-Abschnitt 5-10 Zuweisen des content-Fields

Da ein Dokument mehr als ein Bild enthalten kann, wird die Liste mit den Bildernamen durchlaufen und dem Feld fortlaufend beigefügt (Code-Abschnitt 5-11). Intern wird Lucene ein Array dieses Feldes anlegen, wobei es bei einer Abfrage später möglich sein wird, alle Bildernamen abzurufen.

```
Iterator<String> i = pictureNames.iterator();
while (i.hasNext())
{
        String imagename = i.next();
        Field image = new Field("imagename", imagename,
                  Field.Store.YES,
                  Field.Index.UN_TOKENIZED);
        doc.add(image);
}
```

Code-Abschnitt 5-11 Zuweisen des imagename-Fields

Sobald der `Document`-Instanz alle Felder hinzugefügt worden sind, kann es mit der Indizierung losgehen. Wie bereits gesehen, erfolgt dieser Schritt mit Hilfe der Methode `addDocument` des `IndexWriters`. Nachdem die Indizierung abgeschlossen wurde, kann der Index optimiert werden (Code-Abschnitt 5-12).

Da der Index unoptimiert bei jeder Indizierung ein neues File erhält, kann das Index-Verzeichnis nach einer Weile sehr viele Index-Dateien enthalten. Geht eine spätere Suche über mehere Index-Dateien, wird dadurch deutlich an Performance eingebüsst, da der `IndexWriter` jeweils mehrere Files öffnen und daraus lesen muss.[50]

Aus diesem Grund habe ich mich dazu entschlossen, nach jeder Indizierung gleich eine Optimierung vorzunehmen. Die Aufgabe dieser Optimierung ist es, die verschiedenen Index-Files zu einem einzigen zu verschmelzen.

```
indexWriter.addDocument(doc);
indexWriter.optimize();
```

Code-Abschnitt 5-12 Optimierung des Index

Werden mehrere Dokumente gleichzeitig indiziert (z.B. bei der Indizierung eines Server-Verzeichnisses), sollte darauf geachtet werden, dass die Methode `optimize` erst nach der Indizierung sämtlicher Dokumente aufgerufen wird, da es ziemlich viel Zeit kostet, den Index zu optimieren. Sollte diese Regel nicht befolgt werden, kann sich die Indizierung um einen x-fachen Faktor in die Länge ziehen, was natürlich nicht wünschenswert ist.[51]

Für die Lösung dieses Problems habe ich vorgesehen, dass die Klasse `StandardFileIndexer` einen zweiten Konstruktor mit einem Parameter einer Liste von `InformationToIndex`-Objekten entgegennimmt. Dadurch können zuerst alle Dokumente indiziert werden, bevor der Index optimiert wird. Dieser Prozess wird im Code-Abschnitt 5-13 dargestellt.

[50] Quelle: [Gospodnetic, Hatcher (2005)], Seite 56

[51] Quelle: [Gospodnetic, Hatcher (2005)], Seiten 56, 57, 58

```
Iterator<InformationToIndex> i = itis.iterator();
 ….
while (i.hasNext())
{
       indexWriter.addDocument(doc);
}
indexWriter.optimize();
```

Code-Abschnitt 5-13 Optimierung des Index bei der Indizierung meherer Dokumente

Eventuelle Probleme mit dem Schreibschutz-Mechanismus

Normalerweise wird der `IndexWriter` nach Beendigung der Indizierung über die Methode `close` geschlossen. Da ich in meiner Applikation aber nur über einen einzigen IndexWriter verfüge, kann dieser nicht geschlossen werden. Würde man den IndexWriter schliessen, so wäre das `IndexWriter`-Objekt anschliessend null. Da heisst, es müsste nach jeder Indizierung wieder eine neue `IndexWriter`-Instanz erzeugt werden. Diese Tatsache würde wiederum eine mögliche Inkonsistenz des Index mit sich bringen. Aufgrund dieser Begebenheit wird das `IndexWriter`-Objekt nie geschlossen.

Sobald der `IndexWriter` eine Modifizierung am Index vornimmt, wird im Index-Verzeichnis automatisch (von Lucene selbst) das File *write.lock* angelegt. Ist dieses File im Verzeichnis vorhanden, sollte es für eine aussenstehende Lucene Applikation unmöglich sein, diesen Index zu modifizieren. Da das File direkt mit einer Instanz-des `IndexWriters` verknüpft ist, kann es vorkommen, dass nach einem Neustart oder Absturz des Applikations-Servers dieses *write.lock*-File immer noch vorhanden ist. Startet man die Applikation wieder, kann keine neue Instanz des `IndexWriters` angelegt werden, weil das eben erwähnte File noch existiert und mit einer anderen Instanz des `IndexWriters` verknüpft ist. Ist dies der Fall, werden eine sogenannte `LockObtainFailedException` und eine daraus resultierende `NullpointerException` bei der Indizierung eines Dokuments ausgelöst. Die einzige Lösung hierfür ist das manuelle Löschen des *write.lock*-Files im Index-Verzeichnis.[52]

[52] Quelle: [Gospodnetic, Hatcher (2005)], Seiten 62, 63, 64, 65, 66, 67

6 Suche mit Lucene

Ziel einer Indizierung ist, wie Sie sicherlich bereits wissen, Informationen für eine schnelle Suche bereitzustellen. In diesem Kapitel soll daher die Suche genauer besprochen werden.

6.1 Hauptklassen für die Suche[53]

6.1.1 IndexSearcher

Der `IndexSearcher` bezogen auf die Suche kann mit dem `IndexWriter` bezogen auf die Indizierung verglichen werden: Die zentrale Verbindung zum Index. Die Klasse `IndexSearcher` öffnet den Index in einem Read-Only-Modus und bietet verschiedene Such-Methoden, von denen manche bereits in der Superklasse `Searcher` implementiert sind.

6.1.2 Query

Konkrete Sub-Klassen von `Query` sind zuständig, die Logik für einen bestimmten `Query`-Typ zu kapseln. Instanzen der `Query`-Klasse werden einer Such-Methode des `IndexSearchers` als Argument übergeben. Die einfachste Unter-Klasse von `Query` stellt `TermQuery` dar, deren Zweck es ist, nach einem bestimmten Wort im Index zu suchen. Es existieren zahlreiche andere bereits von Lucene implementierte Query-Subklassen wie `BooleanQuery`, `PhraseQuery`, `PrefixQuery`, `PhrasePrefixQuery`, `RangeQuery`, `FilteredQuery` und `SpanQuery`, auf welche ich jedoch nicht näher eingehen möchte. Im Rahmen meiner Bachelor-Arbeit instanziiere ich nie selber ein solche Klasse, da dies die Klasse `QueryParser` (nächster Abschnitt) für mich übernimmt. Sollten die von Lucene bereits angebotenen `Query`-Implementierungen nicht ausreichen, ist es natürlich möglich, selber eine von `Query` erbende Klasse zu kreieren, die den gewünschten Anforderungen entspricht.

6.1.3 QueryParser

Obwohl die ihm vorigen Abschnitt erwähnten `Query`-Typen sehr Leistungs-stark sein können, ist es nicht unbedingt immer vernünftig, diese im Java-Code selber zu instanziieren. Beim Gebrauch von menschlich-lesbaren textuellen Query-Repräsentationen (wie in meinem Fall) kann der `QueryParser` die zuständigen `Query`-Typen selber konstruieren. Die konstruierte `Query`-Instanz kann sowohl

[53] Quelle: [Gospodnetic, Hatcher (2005)], Seiten 22, 23, 24, 76, 77

eine komplexe Entität, welche aus verschachtelten `BooleanQuerys` und einer Kombination von fast allen erwähnten `Query`-Typen besteht, wie auch eine ganz simple `TermQuery` darstellen.

6.1.4 Hits

Die `Hits`-Klasse ist nichts anderes als ein simpler Container mit Pointern zu den Such-Resultaten. Bei der Suche werden alle Dokumente die mit der Such-Abfrage übereinstimmen, in ein `Query`-Objekt integriert. Aus Performance-Gründen laden `Hits`-Instanzen nur einen kleinen Teil der Resultate auf einmal (Lazy-Loading).

6.2 Beispiel einer einfachen Suche

Ähnlich wie bereits beim Beispiel der Lucene-Indizierung (Kapitel 5.2.2) soll das nachfolgende Beispiel nun im Groben vermitteln, wie die Suche mit Lucene funktioniert.

Um überhaupt eine Suche ausführen zu können, muss selbstverständlich der Standort des entsprechenden Index bekannt sein. Für diesen Zweck wird ein neues `Directory`-Objekt erzeugt. Dies wird im Code-Abschnitt 6-1 dargestellt

```
Directory d = FSDirectory.getDirectory("C:\\temp\\tesindex");
```

Code-Abschnitt 6-1 Erzeugen eines Directory-Objekts

Der nächste Schritt ist die Instanziierung der Klasse `IndexSearcher`, die als Haupt-Akteur für die Suche fungiert. Als Konstruktor-Parameter erhält sie das eben erzeugte `Directory`-Objekt (Code-Abschnitt 6-2).

```
IndexSearcher isearcher = new IndexSearcher(d);
```

Code-Abschnitt 6-2 Erzeugen eines IndexSearcher-Objekts

Es ist nun an der Zeit, eine `QueryParser`-Instanz ins Leben zu rufen (siehe Code-Abschnitt 6-3), deren Aufgabe ist, mittels ihrer Methode parse, ein `Query`-Objekt zu liefern.

Der Konstruktor der Klasse `QueryParser` zwingt uns, ihm eine `Field`- sowie eine `Analyzer`-Instanz mitzugeben. Das `Field`-Objekt korrespondiert dabei zum Datenteil, indem die Suche stattfinden soll. Zu beachten ist dabei, dass das Feld bei der Indizierung angelegt und nicht mit der Option `Field.Index.NO` initialisiert wurde. Das Argument des Typs `Analyzer` bewerkstelligt bei der Suche die Konvertierung des `Query`-Strings in eine optimale und relevante Form (siehe Kapitel 5.2.1.3). Meistens ist es die beste Wahl, bei der Suche denselben Typ des `Analyzers` zu wählen wie bei der Indizierung.

```
String fieldToQuery = „content“;
Analyzer a = new StandardAnalyzer();
QueryParser parser = new QueryParser(fieldToQuery, a);
```

Code-Abschnitt 6-3 Erzeugen eines QueryPqrser-Objekts

Sobald der `QueryParser` erzeugt wurde, ist es dessen Aufgabe, ein spezifisches `Query`-Objekt anhand des Such-Strings zu erzeugen und als Return-Wert zurückzugeben (Code-Abschnitt 6-4 Erzeugen eines Query-Objekts). Für eine optimale Suche wird dabei unser Such-String “Lucene in Action” mittels des `StandardAnalyzers` zum Ausdruck “lucene action” konvertiert.

```
String searchString = „Lucene in Action“;
Query query = parser.parse(searchString);
```

Code-Abschnitt 6-4 Erzeugen eines Query-Objekts

Dieses `Query`-Objekt wird danach der Methode `search` des bereits erzeugten `IndexSearcher`-Objekts übergeben, um die Suche auszuführen. Die Methode `search` liefert dabei ein `Hits`-Objekt, welches alle Such-Resultate enthält (Code-Abschnitt 6-5).

```
Hits hits = isearcher.search(query);
```

Code-Abschnitt 6-5 Ausführen einer Suche

Um nun die einzelnen Dokumente zu erhalten, welche von Lucene als Resultat identifiziert wurden, muss das `Hits`-Objekt durch-iteriert werden. Anand des jeweils erhaltenen `Documents` lässt sich augenblicklich auf ein gewünschtes `Field` zugreifen. Es ist dabei zu beachten, dass das `Field`, auf welches zugegriffen wird bei der Indizierung erzeugt und nicht mit der Option `Field.Store.NO` initialisiert wurde. Die eben erwähnte Option hätte zur Folge (da der Wert des Feldes bei der Indizierung nicht gespeichert wurde), dass das Feld nicht abgerufen werden kann.

```
for (int i = 0; i < hits.length(); i++)
{
      Document doc =  hits.doc(i);
      System.out.println(doc.getField("author").stringValue());
}
```

Code-Abschnitt 6-6 Abfrage der Such-Resultate

In diesem Beispiel (Code-Abschnitt 6-6) wird bei der Iteration durch die Such-Resultate jeweils das Feld *author* abgerufen und auf der Standard-Konsole ausgegeben.

6.3 Lucene-Suche in meiner Applikation

6.3.1 Suche nach Autor und Inhalt

Meine Applikation soll Dokumente finden können, welche der User entweder anhand des Inhalts oder des Autors suchen kann (siehe Abbildung 6-1) . Da bei der Indizierung der Dokumente sowohl der Autor (Feld *author*) wie auch der Inhalt (Feld *content*) in den Index aufgenommen wurden, stellt diese Aufgabe nicht eine allzu grosse Hürde dar.

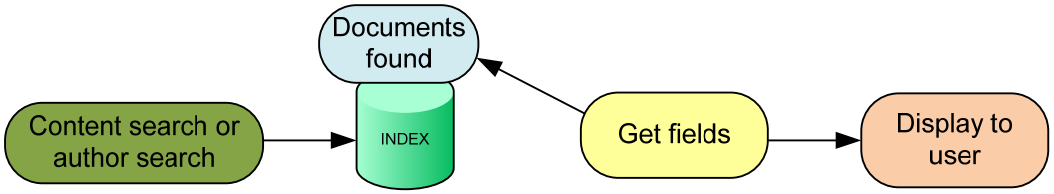


Abbildung 6-1 Suche nach Dokumenten anhand des Inhalts oder des Autors

Für die Suche wurde in der Klasse `TextSearchController` im Paket `td.ivaneggel.jsfmanagedbeans` die Methode *search* beauftragt. Um die Methode möglichst unabhängig zu halten, habe ich mich entschlossen, ihr einen String-Parameter zu geben, der aussagt, in welchem Field die Methode ihre Suche durchführen soll (*author* oder *content*). Nachdem die Suche durchgeführt worden ist, gibt sie eine Liste des Typs `Document` als Return-Wert zurück, welche alle gefundenen Dokumente repräsentiert. Auf die relevantesten Statements bezogen sieht der Body der Methode in etwa folgendermassen aus (Code-Abschnitt 6-7):

```
IndexSearcher isearcher = null;
List<Document> documents = null;

documents = new ArrayList<Document>();
Analyzer a = new StandardAnalyzer();
Directory d =       FSDirectory.getDirectory(ApplicationBean1.
          getCurrentInstance().getIndexLocation());
isearcher = new IndexSearcher(d);
QueryParser parser = new QueryParser(field, a);

Query query = parser.parse(getSearchString());
Hits hits = isearcher.search(query);
for (int i = 0; i < hits.length(); i++) {
       documents.add(hits.doc(i));

 }
```

Code-Abschnitt 6-7 Methode search, welche eine Suche ausführt

6.3.1.1 Klasse SearchResult

Alleine mit den zurückgelieferten Dokumenten der im vorherigen Abschnitt aufgeführten Methode `search` lässt sich natürlich noch nicht viel anstellen. Es ist erforderlich, dass die benötigten `Fields` eines jeden gefundenen `Documents` abgefragt werden, um dem User schlussendlich die gewünschten Informationen zu

liefern. Als Datenhalter dieser Informationen ist die Klasse `SearchResult` mit nachfolgend stehenden Attributen vorgesehen:

- **publicFilePath**
  Relativer Pfad des Dokuments im Web-Kontext. Somit kann später die Datei vom User heruntergeladen werden.
- **filename**
  Datei-Name des Dokuments
- **author**
  Autor des Dokuments
- **text**
  Text des Dokuments
- **abstractText**
  Erste 200 Zeichen des Textes des Dokuments

6.3.1.2 Methode getSearchResult

Die Methode `getSearchResults` in derselben Klasse setzt sich zur Aufgabe, die notwendigen Felder von jedem gefundenen Dokument abzufragen und in ein `SearchResult`-Objekt zu verpacken (siehe Code-Abschnitt 6-8.

```
Iterator<Document> i = results.iterator();
ist<SearchResult> srlist = new ArrayList<SearchResult>();
while (i.hasNext())
{
      Document doc = i.next();
      SearchResult sr = new SearchResult();
      rs.setFilename(doc.getField("filename").stringValue());
      rs.setAbstractText(doc.getField("abstracttext").
          stringValue());
      r.setAuthor(doc.getField("author").stringValue());

      //...
      //etc.
      srlist.add(r);
}
return srlist;
```

Code-Abschnitt 6-8 Methode getSearchResults

Dabei wird jede erzeugte Instanz der Klasse `SearchResult` einer Liste des Typs `SearchResult` hinzugefügt. Am Ende (sobald Schleife durchlaufen) gibt die Methode die Liste als Return-Wert zurück. Anhand dieses Return-Wertes lassen sich später (im Web-Interface) die Resultate mittels einer Tabelle darstellen.

6.3.2 Rolle Lucenes in Verbindung mit Bildern

Die Indizierung und Suche der Bilder spielt sich vollständig über GIFT ab. In einer indirekten Weise ist jedoch Lucene bei der Bereitstellung zusätzlicher

Informationen zuständig. Zusätzliche Informationen sind dann nötig, wenn der User bspw. wissen will, in welchem Dokument das gefundene Bild vorhanden ist. Bei der Indizierung mit Lucene wurden daher bereits alle Bilder-Namen mit-indiziert/persistiert. Aufgrund dieser Tatsache ist es nun möglich, von einem Bild-Namen direkt auf den Dokument-Namen bzw. auf weitere zum Dokument gespeicherte Felder zuzugreifen (siehe Abbildung 6-2). Zusätzlich zu diesem Feature soll der User später ebenfalls alle Bilder eines Dokuments sehen können. Um dies zu gewährleisten ist eine Datenbank-ähnliche Funktion des Index-Feldes (`Field.Store.YES`) für die Abfrage der Bilder-Namen notwendig. Sucht bspw. meine Applikation nach dem File-Namen des Dokuments, kann sie automatisch zu allen im Dokument gespeicherten Datei-Namen der Bilder gelangen (Abbildung 6-3).

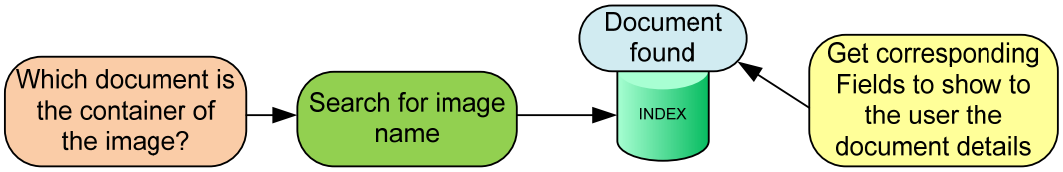


Abbildung 6-2 Welches Dokumente beherbergt ein bestimmtes Bild

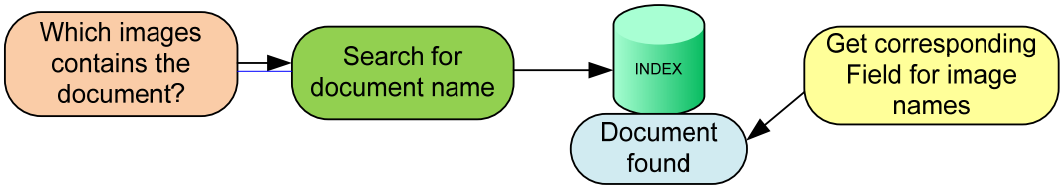


Abbildung 6-3 Welche Bilder gehören zu einem bestimmten Dokument

Der untenstehende Code-Abschnitt 6-9 veranschaulicht eine vereinfachte Version der sich in der Klasse `test4.Search.PicturePreview` befindenden Methode `displayPictures`, welche aufgerufen wird, sobald der User sich die enthaltenen Bilder eines bestimmten Dokuments ansehen will. Der Such-String `(documentName)` wird dabei der Methode `parse` des `QueryParsers` in Anführungszeichen übergeben, so dass über die Methode `search` der Klasse `IndexSearcher` nur das Dokument mit dem exakt gleichen Wert wie dem `documentName` im `Field` *filename* gefunden wird. Sobald das Dokument gefunden wurde, lässt sich über die Methode `getFields` ein Array vom Typ `Field` mit dem Namen *imagename* holen. Da ein Dokument meherere Bilder enthalten kann, wurden bei der Indizierung auch mehrere *imagename*-`Field`s pro Dokument abgespeichert (so viele *imagename*-Felder wie die Anzahl der aus dem Dokument extrahierten Bilder). Aus diesem Grund wird nicht nur ein einziges `Field`, sondern ein Array von Fields verwendet, um alle Bilder-Namen des Dokuments abzufragen. Durch die erhaltenen Felder sind nun alle Bilder-Namen bekannt und lassen sich somit anzeigen.

```
String documentName = this.getParameter();
QueryParser parser = new QueryParser("filename", new
      StanardAnalyzer);

d = FSDirectory.getDirectory(ApplicationBean1.
getCurrentInstance().getIndexLocation());

IndexSearcher isearcher = new IndexSearcher(d);

Query query = parser.parse("\""+documentName+"\"");
Hits hits = isearcher.search(query);
Document doc = hits.doc(0);
Field[] f = doc.getFields("imagename");
```

Code-Abschnitt 6-9 Methode displayPictures

Die Abfrage eines Dokuments anhand eines Bilder-Names soll an dieser Stelle nicht extra aufgeführt werden. Im Prinzip läuft alles mehr oder weniger gleich ab wie im obigen Beispiel. Der grösste Unterschied besteht in der Instanziierung des `QueryParsers`, welcher nun über das Feld *imagename* initialisiert (Code-Abschnitt 6-10) wird (und nicht über *filename* wie vorhin).

```
QueryParser parser = new QueryParser("imagename", a);
```

Code-Abschnitt 6-10 Instanziierung des QueryParsers mit Feld imagename

Sobald die Suche vonstatten gegangen ist, können die relevanten `Fields` für die Anzeige der Dokument-Details (auf der Web-Site) problemlos abgefragt werden. Die Methode die sich dieser Aufgabe stellt trägt den Namen `initFields` und befindet sich in der Klasse `test4.Search.DocumentDetails`.

6.3.3 QueryParser query expressions

Da der vom User im Web-Interface eingegebene Such-String unverändert vom `QueryParser` entsprchend geparst und daraus ein `Query`-Objekt erzeugt wird (nicht programmatisch durch den Programmierer), sollte an dieser Stelle kurz die Möglichkeit der *query expressions* erwähnt werden, welche der `QueryParser` für den User bietet.

Query expressions sind spezielle, die Suche beinflussende Ausdrücke, die im Rahmen einer Such-Abfrage vom `QueryParser` berücksichtigt werden. Die Syntax ist dabei ähnlich wie die von Google zur Suche entwickelte Query-Sprache. Da die gesamte Auflistung aller Features den Rahmen der vorliegenden Arbeit sprengen würde, habe ich mich auf einige Beispiele beschränkt[54] (Tabelle 6-1).

[54] Die Dokumentation für die query expression Syntax finden Sie unter: http://lucene.apache.org/java/2_3_2/queryparsersyntax.html

Query expression	Findet Dokumente die…
`java`	…das Wort *java* im Default-Field enthalten

`java junit`
 	
`java or junit`
	…das Wort *java* oder *junit*, oder beide im Default-Field enthalten.

`+java +junit`
 	
`Java AND junit`
	…die Wörter *java* und *junit* im Default-Field enthalten

`filename:extreme`	…das Wort *extreme* im `filename`-Field enthalten

`filename:extreme`
`-author:ivan`
 	
`Title:extreme`
`AND NOT author:ivan`
	…*extreme* im Feld `filename` enthalten und welche nicht *ivan* im Feld `author` enthalten.

`(agile OR extreme) AND methodology`	…*methodology* und *agile* und/oder *extreme* im Default-Field enthalten
`filename:“extreme.doc“`	..den exakten Ausdruck *extreme.doc* im Feld `filename` enthalten
`Content:“junit action“ ~5`	…die Wörter *junit* und *action* innerhalb von 5 Positionen zueinander enthalten (im Feld `content`).
`java*`	…im Default-Field Wörter mit dem Wort-Stamm *java* enthält. Z.B: *javaserver*, *java.net, javagames*
`java~`	…ähnliche Wörter wie *java* enthalten, z.B. *lava* (im Default-Field)

Tabelle 6-1 Beispiele für query expressions[55]

Anmerkung: in meiner Applikation bezieht sich das Default-Feld auf den Inhalt (*content*) der Dokumente.

[55] Der Inhalt der Tabelle wurde teilweise übernommen aus: [Gospodnetic, Hatcher (2004)], Seite 74

7 Indizierung der Bilder mit GIFT

Für die Indizierung der Bilder ist das Tool GIFT vorgesehen. Wie bereits erwähnt stellt GIFT ein CBIR-System dar. In diesem Kapitel wird beschrieben was ein CBIR ist, welche Funktionalitäten GIFT verwendet und wie sich der Indizierungs-Prozess der Bilder in meiner Applikation abgespielt hat.

7.1 Bedarf an Ablage- und Suchverfahren für Bilder[56]

Der schon fast als dramatisch zu bezeichnende Anstieg von zur Verfügung stehenden Bildern ist heutzutage einer der bedeutendsten Folgen des WWW.

Oftmals unterschätzt wird dabei die damit verbundende Tatsache, dass der Informationsträger Bild innerhalb der ohnehin traditionell auf Bilder ausgerichteten Brachen, aber auch ausserhalb dieser, einem Bedeutungswandel unterzogen wurde und damit fortlaufend an Bedeutung gewinnt. Es ist häufig notwendig, in der ungeheuren Masse, Bilder wieder zu finden, einerseits um diese nicht selber (erneut) erstellen zu müssen, andererseits evtl., um an Informationen zu gelangen, welche mit dem Bild verknüpft sind. Die Kombination aus der Steigerung der Quantität und dem Anstieg der Ansprüche an den Umgang und die Verfügbarkeit mit/von Bildern, haben das Interesse an geeigneten Ablage- und Suchverfahren erheblich erhöht.

7.2 Content-Based Image Retrieval (CBIR)[57]

CBIR stellt die Suche nach Bildern auf der Basis der in jedem einzelnen Bild enthaltenen grafischen Merkmale dar, d.h. die im Bild enthaltenen Informationen werden für die Suche berücksichtigt, nicht die Meta-Daten.

Es lassen sich vier primäre Merkmale eines Bildes feststellen:

---

[56] Quelle: http://www.contentmanager.de/magazin/artikel_218-print_content_based_image_retrieval.html

[57] Quelle: http://www.contentmanager.de/magazin/artikel_218-print_content_based_image_retrieval.html

- **Farbe**: Welche Farbe(n) ist (sind) in einem Bild hauptsächlich vertreten?
- **Farbverteilung**: Wie sind die unterschiedlichen Farben verteilt (Farbhistogramm)?
- **Farbkomposition**: An welcher Position sind welche Farben zu finden?
- **Textur**: Welche Muster oder Konturen erscheinen in einem Bild und wie sind sie im Bild ausgerichtet?

Die unter CBIR zusammengefassten Technologien und Methoden versuchen nach diesen grafischen Merkmalen zu suchen. Dabei werden zwei zentrale Prozesse berücksichtigt:

- Die Indizierung: Dabei wird in einem ersten Schritt der Informationsgehalt der Bilder analysiert und segmentiert. Die in der Bildquelle enthaltenen Informationen (Features) werden anschliessend dem Index beigefügt.
- Die Suche: Die gewonnenen Informationen müssen anschließend auch sinnvoll abgefragt werden können

7.3 GIFT-Funktionalität

7.3.1 Farb-Features[58]

GIFT nutzt eine Palette von 166 Farben, hergeleitet aus der Quantelung des HSV[59] -Raums in 18 Farbtöne, 3 Sättigungs-Grade, 3 Hell-Werte und 4 Grau-Stufen. Dabei werden jeweils 2 Sätze des gequantelten Bildes extrahiert. Der erste ist ein Farb-Histogramm, der zweite repräsentiert das Farb-Layout. Jeder Block des Bildes (wobei der erste das Bild selber ist), wird rekursiv in 4 gleich grosse Blöcke aufgeteilt, in 4 Massstäben. Das Auftreten eines Blocks wird dabei als Binär-Feature betrachtet. Für Bilder der Dimension 256 x 256 gibt es daher 56440 mögliche Farb-Block-Features wobei jedes Bild 340 davon besitzt.

[58] [Squire, Müller, Müller (1999)], Seite 2

[59] Der HSV-Farbraum ist der Farbraum etlicher Farbmodelle, bei denen man die Farbe mit Hilfe des Farbtons (englisch *hue*), der Farbsättigung (*saturation*) und des Hellwerts (bzw. der Dunkelstufe) (*value*) definiert. Quelle: http://de.wikipedia.org/wiki/HSV-Farbraum

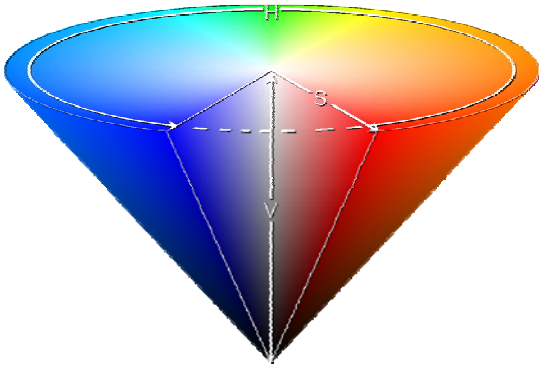


Abbildung 7-1 HSV-Farbraum: Farbton H, *Sättigung* S, *Dunkelstufe* V [60]

7.3.2 Textur-Features[61]

Eine Textur-Suche halt nach visuellen Mustern innerhalb des Bildes Ausschau und berücksichtigt wie diese räumlich definiert sind. An dieser Stelle soll nur erwähnt werden, dass GIFT sogenannte Gabor-Filter anwendet, die Anordnung und Zerlegung der Texturen vorzunehmen. Ein 256 x 256 dimensioniertes Bild hat bei GIFT 3072 mögliche Textur-Features.

7.3.3 Inverted Files[62]

Ein invertiertes File (IF) enthält einen Eintrag für jedes mögliche Feature bestehend aus einer Liste der Items welche das Feature enthalten (Bilder sind einem Feature zugeordnet). Die Text-Retrieval-Community hat Techniken entwickelt, um IFs effizient zu erstellen und zu durchsuchen. Bei der Evaluation einer Query werden dabei nur Bilder zurückgeliefert, welche in der Query präsente Features enthalten.

7.3.4 Gewichtung und Relevanz-Feedback

Das Relevanz-Feedback produziert Queries, welche den Informations-Bedarf des Users besser befriedigt[63]. Durch die Angabe eines Relevanz-Faktors wird die Suche nach Bildern auf diesen angepasst. Dies ist genauer unter Kapitel 8 beschrieben.

---

[60] Quelle: http://upload.wikimedia.org/wikipedia/commons/e/ea/HSV_cone.png

[61] [Squire, Müller, Müller (1999)], Seite 4

[62] [Squire, Müller, Müller (1999)], Seite 4

[63] [Squire, Müller, Müller (1999)], Seite 4

7.4 Prozess der Indizierung in meiner Applikation

Als erstes ist zu erwähnen, dass sich der in meiner Lösung benutzte GIFT-Server nicht extra von mir installiert[64] wurde. Mein Betreuer Dr. Henning Müller hat mir denjenigen der Universität Genf zur Verfügung gestellt. Die Adresse des GIFT-Servers sowie andere sich auf GIFT beziehende Konfigurationen für meine Applikation werden im File *App_Properties.properties* (Beispiel im Anhang) festgelegt.

Ursprünglich war es im Rahmen meiner Bachelor-Arbeit vorgesehen, den Indizierungs-Prozess der Bilder über GIFT automatisch ablaufen zu lassen. Da zum Zeitpunkt der Implementierung meiner Applikation jedoch noch kein Script dafür zur Verfügung stand, hat es nicht mehr gereicht, dieses Feature für meine Lösung zu berücksichtigen.

Die provisorische Lösung der Indizierung der Bilder bestand darin, alle bisher extrahierten Bilder aus dem Verzeichnis *ExtractedImages* zu nehmen, und diese manuell auf dem GIFT-Server zu indizieren. Diese Aufgabe hat freundlicherweise mein Betreuer Dr. Müller für mich vorgenommen. Dabei wurde das ganze Verzeichnis der Bilder in einem einzigen Prozess indiziert.

Da beim Indizierungs-Prozess des Textes und der Meta-Daten mittels Lucene bereits alle Bilder-Namen mit-indiziert worden sind, ist es kein Problem, die Bilder eines Dokuments anzeigen zu lassen. Es ist jedoch zu beachten, dass mit zusätzlichen Dokument-Indizierungen innerhalb meiner Applikation (nach der manuellen Indizierung der Bilder mit GIFT) das System inkonsistent wird. Diese Inkonsistenz ist auf bereits indizierte Dokumente (bezogen auf Text und Meta-Daten) zurückzuführen, deren Bilder noch nicht indiziert wurden. Will der User bspw. ähnliche Bilder zu einem in einem Dokument enthaltenen Bild (mittels GIFT) suchen (durch einen Link-Klick unterhalb des Bildes, siehe dazu Kapitel 9.6.5 und Abbildung 9-7), welches auf dem GIFT-Server noch nicht indiziert (und damit nicht abgespeichert) wurde, kann keine Ähnlichkeits-Suche durchgeführt werden. Für den produktiven Gebrauch meiner Applikation es dringend notwendig, den Vorgang der Indizierung der Bilder zu automatisieren. Dies sollte idealerweise über einen Web-Service geschehen, welcher auf der Maschine des GIFT-Servers läuft.

[64] Download GIFT: ftp://ftp.gnu.org/gnu/gift

8 Visuelle Suche mit GIFT

Nach der Indizierung der Bilder mit GIFT soll an dieser Stelle die dazugehörige Suche vorgestellt werden. Dieses Kapitel setzt sich mit der GIFT-Suche allgemein und im speziellen Fall (meiner Applikation) auseinander.

8.1 Grundlegendes zur Kommunikation mit GIFT

8.1.1 Verbindung zum GIFT-Server

Wie allgemein in einer Client-Server Architektur üblich, hört der GIFT-Server einen bestimmten Port ab. Will sich nun ein Client mit dem GIFT-Server verbinden, muss er dies über den festgelegten Port tun. Sobald der Client mit dem Server verbunden ist, sind beide Parteien bereit, miteinander zu kommunizieren. Von der Programmiersprache Java aus gesehen, wird die Verbindung über ein sogenanntes `Socket` realisiert.

```
Socket socket = new Socket(host, port);
```

Code-Abschnitt 8-1 Erzeugung einer Socket-Instanz

Aus dem obenstehenden Code-Abschnitt 8-1 ist zu entnehmen, wie ein neues `Socket` instanziiert wird. Der Hostname (oder IP-Adresse) und Port des Servers wird dem Konstruktor der `Socket`-Klasse übergeben. Ist die Verbindung einmal hergestellt, lässt sich über das `Socket`-Objekt ein `DataOutputStream`-Objekt holen. Über diesen Stream werden anschliessend Nachrichten zum Server gesendet. Erhält der Server eine Nachricht, verarbeitet er diese und sendet eine Nachricht an den Client zurück. Über die Methode `getInputStream` der `Socket`-Instanz kann der Client schliesslich die gesendete Message des Servers empfangen.

8.1.2 Protokoll MRML[65] [66]

Das Format der gesendeten und empfangenen Nachrichten heisst MRML, die Multimedia Retrieval Markup Language, welche von der Universität Genf entwickelt wurde. Die XML basierte Sprache MRML setzt sich als Ziel, den Zugang zu Multimedia-Abruf- und –Management-Software zu vereinheitlichen,

[65] Quelle: http://www.mrml.net., Stand 02.10.2008)

[66] Die Dokumentation von MRML findet sich unter: http://www.mrml.net/specification/doc/index.html

um deren Leistungsfähigkeit zu verbessern. Um einmal in Groben Zügen zu sehen, wie in etwa sich die Kommunikation über MRML abspielt, habe ich zwei Beispiele dazu aufgeführt.

8.1.2.1 Beispiel für den Empfang zufälliger Bilder[67]

Will ein Client bspw. 2 zufällige Bilder des GIFT-Servers empfangen, muss er dazu folgende MRML-Nachrichten verfassen und an den Server senden.

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE mrml SYSTEM
"http://isrpc85.epfl.ch/Charmer/code/mrml.dtd">
<mrml session-id="48">
<query-step  session-id="48" result-size="5" algorithm-
id="adefault" collection="c-37-50-13-30-8-108-2-273-0"/>
</mrml>
```

Code-Abschnitt 8-2 MRML-Dokument für den Empfang zufälliger Bilder (vom Client zum Server)

Anhand des Code-Abschnitt 8-2 MRML-Dokument für den Empfang zufälliger Bilder lässt sich feststellen, dass dazu verschiedene Angaben an den Server übermittelt werden müssen:

- **session-id:**
  Jede Verbindung zum GIFT-Server benötigt eine Session-ID. Diese Session-ID identifiziert die Verbindung und damit gleichzeitig den Client. Erhält der Server eine Anfrage mit der Session-ID 57, sendet er eine Antwort an die Verbindung mir der ID 57 zurück. Bei der Verbindungs-Herstellung (Server-Handshake) wird dem Client die ID mitgeteilt.
- **result-size:**
  Repräsentiert die vom Client gewünschte Anzahl an Resultaten.
- **algorithm-id:**
  Es existieren verschiedene Algorithmen zur Berechnung der Ähnlichkeit von Bildern, daher muss bei jedem Request die ID des Algorithmus mitgegeben werden. In meiner Bachelor-Arbeit verwende ich ausschliesslich die ID *adefault* (*separate normalization*).
- **collection:**
  Der GIFT-Server beherbergt eine oder mehrere Collections. Einfach umschrieben stellt eine solche Collection eine Sammlung von auf dem GIFT-Server indizierten und gleichzeitig abgespeicherten Bildern dar. Will ein GIFT-Client auf dem dazugehörigen Server eine Operation ausführen, bezieht sich diese immer auf eine bestimmte Collection.

Unverzüglich nach dem Erhalt und der anschliessenden Verarbeitung der Message, sendet der GIFT-Server eine Nachricht im Stil der nachfolgenden (Code-Abschnitt 8-3) zurück.

[67] Dieser Abschnitt bezieht sich auf eigene Recherchen im Source- Code des SnakeCharmers

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<mrml  just-for-test="and-of-course-for-fun" session-id="51" >
  <cui-time-stamp  calendar-time="Thu Oct 23 09:25:46 2008"/>
  <query-result>
    <query-result-element-list>
      <query-result-element  calculated-similarity="1.000000"
        image-location="http://medgift.unige.ch/
I       mages/hesso/bsp1.jpg"
        thumbnail-location="httpblabla " />

      <query-result-element  calculated-similarity="1.000000"
        image-location="http://medgift.unige.ch/
        images/hesso/bsp2.jpg" thumbnail-
        location="http://medgift.unige.ch/
        images/hesso_thumbnails/bsp2_JPG.jpg"/>
      </query-result-element-list>
  </query-result>
  <cui-time-stamp  calendar-time="Thu Oct 23 09:25:46 2008"/>
</mrml>
```

Code-Abschnitt 8-3 MRML-Dokument für den Empfang zufälliger Bilder (vom Server zum Client)

In der erhaltenen MRML-Message sind folgende erwähnenswerte Tags und Attribute enthalten

- **query-result-element**
  Ein *query-result-element* stellt ein einzelnes vom Server berechnetes Resultat dar.
- **calculated-similarity**
  Stellt die von GIFT berechnete Ähnlichkeit des Bildes relativ zum Abfrage-Bild (können auch mehrere sein) dar. Diese Ähnlichkeit wird in normalisierter Form dargestellt. In diesem Beispiel sind die berechneten Ähnlichkeiten beide 1.0, da wir dem Server keine Query-Bilder mitgegeben haben. Falls dies nicht geschieht, schickt der Server automatisch URLs zufälliger Bilder der Collection zurück.
- **image-location**
  Repräsentiert die URL eines Resultat-Bildes.
- **thumbnail-location**
  Repräsentiert die URL des Thumbnails des Resultat-Bildes.

8.1.2.2 Beispiel für eine Ähnlichkeits-Suche[68]

Für das Empfangen zufälliger Bilder wurden der MRML-Message keine Relevanz-Bilder mitgegeben. Um eine Ähnlichkeits-Suche durchzuführen muss daher dem Server mitgeteilt werden, zu welchem Bild/Bildern er ähnliche Resultate liefern soll.

[68] Dieser Abschnitt bezieht sich auf eigene Recherchen im Source- Code des SnakeCharmers

Die MRML-Message für einen Request sieht in etwa folgendermassen aus (Code-Abschnitt 8-4):

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE mrml SYSTEM
    "http://isrpc85.epfl.ch/Charmer/code/mrml.dtd">
  <mrml session-id="54">
  <query-step session-id="54"  result-size="2"
     result-cutoff="0.0"
     collection-id="c-37-50-13-30-8-108-2-273-0"
     algorithm-id="adefault">
    <user-relevance-element-list>
      <user-relevance-element image-
        location="http://medgift.unige.ch/images
       /hesso/Form_Template_Method_[final](2).ppt_2008_9_30_
       1222766979062_9.PNG" thumbnail-location=""
       user-relevance="1"/>
      <user-relevance-element image-
        location="http://medgift.unige.ch/images
        /hesso/JavaServer.Faces.JSF.in.Action.pdf_
        2008_9_30_1222767362359_216.jpg" thumbnail-location=""
        user-relevance="-1"/>
    </user-relevance-element-list>
  </query-step>
</mrml>
```

Code-Abschnitt 8-4 MRML-Request für Ähnlichkeits-Suche

Im obigen Beispiel (Code-Abschnitt 8-4) wird eine Request-MRML-Message verfasst, welche 2 Bilder ähnlich zu einem relevanten und möglichst unähnlich zu einem irrelevanten Bild sucht.

Nachfolgend die wichtigsten Tags und Attribute für eine Ähnlichkeits-Suche:

- **user-relevance-element-list**
  In diesem Tag werden alle relevanten Elemente platziert.
- **user-relevance-element**
  Dies ist die Einheit für ein Abfrage-Bild.
- **image-location**
  Der Wert dieses Attributs beschreibt die URL-Adresse des Bildes, welche in die Such-Abfrage miteinbezogen werden soll. Es ist nicht zwingend dass die URL des Bildes sich auf dem GIFT-Server befindet. Als Resultate hingegen werden nur auf dem GIFT-Server lokalisierte Bilder geliefert.
- **user-relevance**
  Das Attribut *user-relevance* gibt an, wie relevant das Bild in der Such-Abfrage behandelt werden soll (Relevanz-Feedback). Für den Wert des Attributs sind ganzzahlige Werte des positiven wie auch negativen Bereichs erlaubt. Je grösser der Wert für user-relevance gewählt wird, umso relevanter wird das Bild für die Suche ähnlicher Bilder behandelt. Je kleiner Wert, desto irrelevanter ist das Bild. Wird bspw. eine *user-relevance* von -1 gewählt, werden Bilder zurückgeliefert, welche möglichst anders als das Abfrage-Bild aussehen. Eine *user-relevance* von 0 wird neutral behandelt. Werden der MRML-Message mehrere *user-relevance-*

element-Tags mitgegeben, so wird die Relevanz all dieser Elemente berücksichtigt. Falls dem Server ein Element mit der Relevanz *1* und ein anderes mit der Relevanz *-1* übermittelt wird, sucht GIFT Bilder, welche dem ersten Bild ähnlich sehen, jedoch visuell möglichst ungleich dem zweiten Bild sind. Im Rahmen meiner Applikation habe ich nur Bilder mit Relevanz *1* und *-1* für die Ähnlichkeits-Suche berücksichtigt, wobei *1=relevant* und *-1=irrelevant* bedeutet.

8.2 GIFT-Client meiner Applikation

Um meine Web-Applikation als GIFT-Client fungieren zu lassen, habe ich mich, wie bereits erwähnt, auf den Open-Source-Java-GIFT-Client SnakeCharmer von Zoran Pecenovic gestellt. Obwohl SnakeCharmer als Java Applet programmiert wurde, und bereits ca. 10 Jahre alt ist, habe ich Teile davon benutzen können.

8.2.1 Aufbau von SnakeCharmer[69]

Aufgabe des SnakeCharmers ist es, eine Verbindung zum GIFT-Server herzustellen und anschliessend über das Protokoll MRML mit diesem zu kommunizieren. Da der SnakeCharmer selbst sich mit der Generierung nötigen MRML-Codes befasst, war es für mich glücklicherweise nicht erforderlich, mich selbst darum zu kümmern.

8.2.1.1 Herstellen einer Verbindung

Die Klasse `CharmerConnectionMRML` des SnakeCharmers ist die Haupt-Komponente für die Interaktion mit dem GIFT-Server. Die Instanziierung der Klasse erfolgt folgendermassen (Code-Abschnitt 8-5):

```
CharmerConnectionMRML cc = new CharmerConnectionMRML(null,
                    null, serverAddress);
```

Code-Abschnitt 8-5 Instanziierung einer CharmerConnectionMRML

Der Konstruktor des Typs `CharmerConnectionMRML` verlangt drei Parameter (User-Name, Collection und Server-Adresse), wobei meiner Meinung nach (gemäss eigener Recherchen im Source-Code) nur einer davon, nämlich die Server-Adresse, relevant ist. Das Erzeugen eines neuen `CharmerConnectionMRML`-Objekts veranlasst den Client zu einem Handshake mit dem GIFT-Server, wobei der Client die Session-ID erhält. Sobald das instanziierte Objekt die benötige Session-ID besitzt, ist der Client in der Lage, Operationen auf dem Server ausführen zu lassen.

[69] Dieser Abschnitt bezieht sich auf eigene Recherchen im Source- Code des SnakeCharmers

8.2.2 Empfangen zufälliger Bilder[70]

Wie bereits vorhin gesehen, lassen sich eine gewünschte Anzahl zufälliger Bilder-URLs vom Server empfangen. Um diese Aufgabe durchzuführen, steht die Methode `getRandomImages` der Klasse `CharmerConnectionMRML` bereit. Gemäss MRML-Definition muss für den Empfang der zufälligen Bilder die MRML-Message mit diversen Informationen (Anzahl Resultate, Alogorithmus-ID, Collection) versehen werden. Genau aus diesem Grund besitzt die Instanz-Methode eine Parameter-Liste für die eben erwähnten Informationen. Der letzte Parameter der Methode ist Java-Applet-spezifisch und daher für meine Applikation irrelevant. Angelehnt an diese Parameter leitet die Methode `getRandomImages` die Generierung des MRML-Codes mit der durch den Server-Handshake erhalteten Session-ID ein. Nach dem Verfassen der MRML-Nachricht wird diese über einen `OutputStream` an den GIFT-Server übergeben. Das unten aufgeführte Beispiel (Code-Abschnitt 8-6) zeigt den Aufruf von `getRandomImages` über das vorhin erzeugte `CharmerConnectionMRML`-Objekt.

```
cc.getRandomImages(5, "adefault", "c-37-50-13-30-8-108-2-273-
0", null);
```

Code-Abschnitt 8-6 Aufruf der Methode getRandomImages

Beim Aufruf der Methode `getRandomImages` wird die konsekutive MRML-Message (Code-Abschnitt 8-7) verfasst und dem Server übermittelt.

```
<mrml session-id="57">
<query-step  session-id="57" result-size="5" algorithm-
     id="adefault" collection="c-37-50-13-30-8-108-2-273-0"/>
</mrml>
```

Code-Abschnitt 8-7 Generierte MRML-Message durch die Methode getRandomImages

Wie bereits thematisiert ist es nun die Aufgabe des Servers, die ihm zugekommene MRML-Nachricht zu parsen und aus den daraus gewonnen Informationen die vom Client gewünschten Operationen auszuführen. Sind Server-seitig die geforderten Prozeduren erfolgt, wird das Resultat vom Server in eine MRML-Message gepackt und an einen `OutputStream` weitergeleitet. Der Client ist nun bereit die MRML-Nachricht über einen `InputSream` entgegenzunehmen. Die Methode `getRandomImages` kümmert sich zusätzlich um dieses Unterfangen. Ist die Server-Antwort einmal empfangen wird die MRML-Message mit dem XML-Parser SAX von Microstar geparst, die daraus extrahierten Informationen in `QueryObjects` eingespeist und als Return-Wert (`QueryObject`-Array) der Methode `getRandomImages` zurückgeliefert.

[70] Dieser Abschnitt bezieht sich auf eigene Recherchen im Source- Code des SnakeCharmers

8.2.3 QueryObject-Klasse[71]

Die `QueryObject` Klasse ist ein reiner Datenhalter. In einer Instanz dieser Klasse werden nur Informationen für/von eine(r) Query abgestellt (siehe Code-Abschnitt 8-8). Die wichtigsten Attribute der Klasse sind:

- **location:**
  Entspricht der URL des abzufragenden bzw. erhaltenen Bildes.
- **mThumbnailLocation:**
  Entspricht der URL des Thumbnails des erhaltenen Bildes.
- **type:**
  Stellt die Relevanz eines Bildes für eine Query / von einer Query dar.

```
QueryObject[] result = cc.getRandomImages(5, "adefault", "c-37-
50-13-30-8-108-2-273-0", null);
```

Code-Abschnitt 8-8 Zuweisung eine QueryObjects durch die Methode getRandomImages

Da das `QueryObject`-Array `result` nun alle Resultate gespeichert hat, können diese problemlos mit einem Schleifen-Durchlauf geholt werden. In Bezug auf meine Web-Applikation wird das Attribut `location` benutzt, um daraus an den File-Namen der Resultat-Bilder zu gelangen. Da die Bilder und Thumbnails auch auf meinem eigenen Server gespeichert sind, kann ich diese anzeigen lassen (es wäre auch möglich URL der Bilder auf dem GIFT-Server zu benutzen).

8.2.4 Ähnlichkeits-Suche[72]

Für die Durchführung einer Ähnlichkeits-Suche steht die Methode `doSimilaritySearch` der Klasse `CharmerConnectionMRML` zur Verfügung. Aufgabe dieser Methode ist es wiederum, die für die Konstruktion der MRML-Message nötigen Parameter entgegenzunehmen, die MRML-Nachricht anhand der erhaltenen Parameter zu konstruieren und anschliessend zum GIFT-Server zu übermitteln. Im Anschluss daran wartet die Methode mit Hilfe eines `InputStreams` des Sockets bis sie die zurückgesendete Resultat-Message des Servers erhält. Als Return-Wert liefert die Methode erneut ein Array des Typs QueryObject.

Als Parameter erhält die Methode `doSimilaritySearch` ein `Query`-Object-Array, die ID des zu verwendenen Algorithmus, sowie die Collection, auf welche sich der GIFT-Server beziehen soll (siehe Code-Abschnitt 8-12).

[71] Dieser Abschnitt bezieht sich auf eigene Recherchen im Source- Code des SnakeCharmers

[72] Dieser Abschnitt bezieht sich auf eigene Recherchen im Source- Code des SnakeCharmers

```
QueryObject qo1 = new QueryObject();
qo1.location = new URL("http://medgift.unige.ch/images/hesso/
       Form_Template_Method_[final](2).
       ppt_2008_9_30_1222766979062_9.PNG");
qo1.type = 1;
```

Code-Abschnitt 8-9 Erzeugung eines Query Objects

Für die Kreation eines `QueryObject`-Objekts wird der Standard-Konstruktor verwendet (Code-Abschnitt 8-10). Da die Attribute der `QueryObject`-Klasse alle öffentlich zugänglich sind (*public-accessor*), können ihnen die Werte direkt zugewiesen werden. Die für die Ähnlichkeit belangvollen Attribute sind `location` und `type`. Das Attribut `type` ist wichtig, weil durch dessen Wert die Relevanz des Bildes festgelegt wird. In obigem Beispiel wird dem QueryObject `qo1` *1* (für relevant) als `type` zugeordnet.

Es ist durchaus denkbar, mehrere `QueryObject`-Objekte zu erzeugen (Code-Abschnitt 8-10). Sobald alle nötigen Objekte mit Daten gefüllt wurden, werden diese zu einem Array zusammengefasst (Code-Abschnitt 8-11). Wird dieses Array nun der Methode `doSimilaritySearch` übergeben (Code-Abschnitt 8-12), werden alle `QueryObject`-Objekte und damit ihre Relevanz (oder Irrelevanz) berücksichtigt.

```
QueryObject qo2 = new QueryObject();
qo2.location = new
URL("http://medgift.unige.ch/images/hesso/JavaServer.Faces.JSF.
in.Action.pdf_2008_9_30_1222767362359_216.jpg");
qo2.type = -1;
```

Code-Abschnitt 8-10 Erzeugung eines QueryObjects

```
QueryObject[] query = new QueryObject[2];
query[0] = qo1;
query[1] = qo2;
```

Code-Abschnitt 8-11 Erzeugung eines QueryObject-Array

```
QueryObject[] result = cc.doSimilaritySearch(query, 5,
"adefault", "c-37-50-13-30-8-108-2-273-0", null);
```

Code-Abschnitt 8-12 Aufruf der Methode doSimilaritySearch

Sobald der Wert des `QueryObject` Arrays `result` durch die Methode `doSimilaritySearch` zugewiesen wurde, können wiederum die Attribute eines jeden `QueryObjects` abgerufen und gegebenenfalls angezeigt werden.

8.2.5 Klasse GIFTSearcher

Um die in meiner Applikation verwendeten Methoden der Klasse `CharmerConnectionMRML` des Programm SnakeCharmer ein wenig zu kapseln, habe ich die Klasse `td.ivaneggel.searching.gift.GIFTSearcher`

implementiert. Gemäss Code-Abschnitt 8-13 sieht diese folgendermassen aus (nur die wichtigen Methoden sind aufgeführt):

```
public class Giftsearcher
{

    CharmerConnectionMRML cc;
    public void connect()
    {
       cc  = new CharmerConnectionMRML(getGiftUser(),
          getDefaultCollection(),
          getGiftIp()+":"+getGiftPort());
    }

    public void close()
    {
        if (cc!=null)cc.close();
    }

    public QueryObject[] getRandomImages(int number)
    {
        QueryObject[] qo = cc.getRandomImages(number,
                this.getDefaultAlgorithm(),
                getDefaultCollection(), null);
        return qo;
    }

    public QueryObject[] doSimilaritySearch(QueryObject[] qo,
         int maxReturnImages)
    {
        return cc.doSimilaritySearch(qo, maxReturnImages,
                getDefaultAlgorithm(),
                getDefaultCollection(), null);
    }
}
```

Code-Abschnitt 8-13 Klasse GiftSearcher

Aus dem Code (Code-Abschnitt 8-13) ist zu entnehmen, dass darin 4 wichtige Methoden enthalten sind. Zum einen ist es die Methode `connect`, welche eine Verbindung zum GIFT-Server (Handshake) durchführt. Für die Schliessung der Verbindung ist die Methode `close` vorgesehen. Damit werden speicherlastige Ressourcen wie der `InputStream` und der `OutputStream` des `Sockets` wieder freigegeben.

Die beiden restlichen Methoden `getRandomImages` und `doSimilaritySearch` sind im Prinzip nichts anderes als Delegate-Methoden, welche die zugehörigen Methoden der Klasse `CharmerConnectionMRML` aufrufen.

9 Web Interface

Wie bereits angesprochen wurde des Vorteils einer Client-seitigen Plattform- und Standort-Unabhängigkeit wegen die User-Schnittstelle als Web-Interface realisiert. Dieses Kapitel beschreibt die Einbindung der in den vorderen Kapiteln beschriebenen Indizierung und Suche in eine Web-Oberfläche (JSF).

9.1 Aufbau von JSF

Um einen kurzen Einblick in die Technologie JSF zu geben, wird im Folgenden kurz und knapp der Aufbau von JSF erläutert.

JavaServer Faces bietet kurz gesagt eine strikte Trennung der Darstellung von der Geschäfts-Logik. Um das Layout einer Page kümmert sich immer ein sogenanntes JSP-Dokument. Von diesem JSP File aus ist es möglich, Methoden eines sogenannten Managed-Beans, welches Java-Code beherbergt, aufzurufen.

9.1.1 Managed-Beans

Ein Managed-Bean ist im Prinzip nichts anderes als eine ganz normale Java-Klasse. In so einem Bean ist es möglich, sowohl Instanz-Variablen wie auch Methoden zu halten. Im Code ist es erlaubt, andere verfügbare Java-Klassen zu instanziieren , wie man es sich auch sonst von Java gewöhnt ist. Der einzige Unterschied eines Managed-Beans zu einer herkömmlichen Java-Klasse ist die Koppelung an JSF. Von einer JSP-Seite aus ist der Programmierer im Stande, gewisse öffentliche Methoden eines Managed-Beans aufzurufen. Diese Koppelung wird durch einen Eintrag im Konfigurations-File *faces-config.xml* erreicht.

9.1.1.1 Konfigurations-Datei faces-config.xml

Jedes Managed-Bean muss in ein Konfigurations-File namens faces-config.xml (siehe Code-Abschnitt 9-1), welches sich im *WEB-INF*-Ordner befindet, eingetragen werden. Der Bereich eines Managed-Beans lässt sich mit einem von drei zur Auswahl-stehenden Optionen belegen. Folgende sind definiert:

- **Request:**
  Ist der Scope eines Beans als *request* gewählt, wird das Managed-Bean bei jedem Request des Users neu instanziiert.
- **Session**
  Beim Session-Scope wird die Managed-Bean-Klasse nur einmal für jede HTTP-Session instanziiert. Instanzvariabeln stehen damit während der gesamten Dauer der Session zur Verfügung.
- **Application**
  Der Application-Scope wird benötigt, um Applikations-spezifische Ressourcen bereitzustellen. Das Managed-Bean wird nur einmal (beim

Start der Applikation) instanziiert und bleibt bis zur Terminierung der Web-Applikation ständig verfügbar.

```
<managed-bean>
   <managed-bean-name>
      sampleBean
   </managed-bean-name>
   <managed-bean-class>
      td.ivaneggel.jsfmanagedbeans.SampleBean
   </managed-bean-class>
   <managed-bean-scope>
      request
   </managed-bean-scope>
</managed-bean>
```

Code-Abschnitt 9-1 faces-config.xml

9.1.2 JSP-Seite

Eine JSP-Seite ist ein XML-basiertes Dokument (normalerweise), welches für das Layout der später angezeigten Web-Site zuständig ist. Dieses JSP-File wird mit sogenannten Komponenten-Bäumen realisiert. Eine JSF-Komponente beschreibt eine Klasse, welche zuständig für die Generierung von HTML- und JavaScript-Code ist. Solch eine Komponente wird in deklarativem Stil in eine JSP-Page eingebunden. Die einzelnen Komponenten werden hinterher bei einem Request (des Web-Anwenders) vom Framework JSF automatisch instanziiert und können anschliessend Java-Methoden bestimmter Klassen, den Managed-Beans, aufrufen. Auch ist es möglich, in Java-Klassen Zugang zu den erzeugten Komponenten-Objekten zu erlangen.

9.1.3 Kurzes Beispiel

Das nachfolgende kurze Beispiel soll nur einen sehr simplen Anwendungsfall in JSF darstellen.

```
<h:commandButton value=”test”
action=”#{sampleBean.button_action}”/>
<h:outputText value=”#{sampleBean.output}”/>
```

Code-Abschnitt 9-2 Einträge im JSP-File

```
public class SampleBean
{

    private String output = “”;
    public String button_action()
    {
        output = “Hello World”
        return null;
    }

}
```

Code-Abschnitt 9-3 Code im Managed-Bean

Im JSP-File wird ein simpler Button über die Komponente `h:commandButton` erzeugt. Als Action-Event wird mit Hilfe der *Java-Expression-Language* die Methode des ManagedBeans `sampleBean` registriert (Code-Abschnitt 9-2). In dieser Methode wird nun die String-Instanz-Variable `output` auf den Wert *Hello World* festgelegt. Die zweite Komponente, `outputText`, ist für die Anzeige dynamischen Textes konzipiert. Der String `output` in der Managed-Bean-Klasse `sampleBean` definiert den Anzeige-Wert dieses Elements (Code-Abschnitt 9-3). Durch Anklicken des Buttons durch den User wird der Text *Hello World* auf der Seite ausgegeben.

9.2 Indizierung eines hochgeladenen Dokuments

Wie sie bereits aus diesem Dokument entnommen haben, besteht für den End-User die Möglichkeit, ein hochgeladenes Dokument indizieren zu lassen.

Damit der User ein Dokument hochladen kann, wird die JSF-Komponente `webuijsf:upload` in das JSP Dokument eingebunden (Code-Abschnitt 9-4). Diese Komponente wird später zur Laufzeit ihrerseits das HTML-Element *input file* generieren. Um Zugriff auf die Upload Komponente im Java-Code zu haben, wird dazu ein sogenanntes Binding zu einer Upload-Komponent im Managed-Bean `IndexUploadedFile` hergestellt.

```
<webuijsf:upload
binding="#{Index$IndexUploadedFile.fileUpload1}"
id="fileUpload1"/>
<h:commandButton
action="#{Index$IndexUploadedFile.button1_action}"/>
```

Code-Abschnitt 9-4 Upload-und Button-Komponente im JSP-File

Zusätzlich soll ein Button mit dem Tag `h:commandButton` deklariert werden. Sobald der User diesen Button klickt, wird ein Submit an den Server ausgeführt. Dabei wird die Methode `button1_action` im JSF-Managed-Bean `IndexUploadedFile` aufgerufen. Die Aufgabe der Methode ist es, den Upload, die Extraktion und die Indizierung des Dokuments auszuführen.

9.2.1 Upload

```
UploadedFile uploadedFile = fileUpload1.getUploadedFile();

String fileName =
UtilClass.getFilename(uploadedFile.getOriginalName());
```

Code-Abschnitt 9-5 Zuweisung des Namens des upzuloadenden Files

Das upzuloadende File wird mit Hilfe der Methode `getUploadedFile` über das Objekt `fileUpload1` geholt (Code-Abschnitt 9-5), welches an die im JSP-File definierte Komponente gebunden ist, d.h. es wird erzeugt, sobald der User einen Request an die Seite fordert. Die Validierung und die Speicherung erfordert den File-Namen des Dokuments, welcher dem String *fileName* zugeordnet wird.

Gemäss der unter Kapitel 3.2.4 beschriebenen Prozedur erfolgt nun die Validierung und die Speicherung des Files auf der Festplatte. Da dieses Verfahren schon detailliert erklärt wurde, möchte ich darauf nicht mehr näher eingehen.

9.2.2 Extraktion

Für die Extraktion des hochgeladenen Dokuments wird der durch den erfolgten Upload bereits bekannte dazugehörige Pfad verwendet, um ein Objekt der Klasse `ExtractedDocument` zu erzeugen (Code-Abschnitt 9-6).

```
ExtractedDocument ed =
FileEndingChecker.getInstance(uploadedDocumentFileAbsolutePath);
```

Code-Abschnitt 9-6 Instanziierung der Klasse ExtrcatedDocument

Nach der Instanziierung wird dieses Objekt der Methode `extractAndIndex` übergeben, welche, wie der Name schon vermuten lässt, sich um die Extraktion und Indizierung des Dokuments kümmert. Die Extraktion läuft dabei nach dem gleichen Schema ab, wie bereits unter Kapitel 4 gesehen: Alle benötigten Informationen lassen sich bequem über das `ExtractedDocument`-Objekt `ed` beschaffen (Code-Abschnitt 9-7).

```
author = ed.getAuthor();
publicfileaddress = this.getApplicationBean1().
             getRelativePathUploadedDocumentsDirectory()+
            "/"+fileName);
content = ed.getText();
ed.saveImages(imageDirectoryAbsolutePath,
thumbDirectoryAbsolutePath);
```

Code-Abschnitt 9-7 Zuweisung der benötigten Informationen mit Hilfe des ExtractedDocument-Objekt

9.2.3 Indizierung

Sowie alle erforderlichen Informationen des Dokuments extrahiert sind, sollen diese in eine `InformationToIndex`-Instanz gepackt werden (Code-Abschnitt 9-8).

```
InformationToIndex iti = new
InformationToIndex(publicfileaddress, content, author,
ed.getPictureNames());
```

Code-Abschnitt 9-8 Instanziierung eines InformationToIndex-Objekts

Für die Ausführung der Indizierung muss die Instanz der Methode `indexInformation` der Klasse `StandardFileIndexer` weitergereicht werden. Als zusätzliches Argument der Methode wird das *IndexWriter*-Objekt aus dem Application-Scoped-Managed-Bean `ApplicationBean1` übergeben (Code-Abschnitt 9-9).

```
IndexWriter iw =
          this.getApplicationBean1().getIndexWriter();
StandardFileIndexer sfi = new
          StandardFileIndexer(iw, iti);
sfi.indexInformation();
```

Code-Abschnitt 9-9 Holen des IndexWriters und Aufruf der Methode indexInformation

Der Aufruf der Methode `stf.indexInformation` bewirkt die Indizierung des Textes und der Meta-Informationen (Code-Abschnitt 9-9).

9.3 Indizieren eines im WWW verfügbaren Dokuments

Für die Indizierung eines im Web öffentlich erreichbaren Dokuments ist es erforderlich, dieses zuerst auf den Server herunterzuladen.

9.3.1 Download

Um den Download und damit die anschliessende Extraktion und Indizierung eines Dokuments zu veranlassen, wird User-Input in Form einer URL des im Web erreichbaren Files erwartet. Dies geschieht über ein Text-Feld. Nach der Bestätigung des Users durch den Klick eines Submit-Buttons, soll der Download, die Extraktion und die Indizierung des Dokuments veranlasst werden.

Das JSP-Tag `h:commandButton` ist an das Managed-Bean `IndexPublicFile` gekoppelt (Code-Abschnitt 9-10). Nach Betätigung des Buttons wird die Methode `button1_action` ausgeführt. Diese ist zuständig für die Ausführung der relevanten Operationen.

```
<h:commandButton
  action="#{Index$IndexPublicFile.button1_action}"
  value="Download and Index"/>
<h:inputText
  value="#{Index$IndexPublicFile.path}"/>
```

Code-Abschnitt 9-10 Definition der Text-Feld- und Button-Komponenten

Anhand der *Java-Expression-Language* ist der User-Input des Text-Feldes `h:inputText` mit der String-Variable `path` der Klasse `IndexPublicFile` verknüpft (Code-Abschnitt 9-10). Somit kann die vom User in das Text-Feld eingegebene URL problemlos in der Methode `button1_action` abgefragt werden. Mittels dieses Pfades erfolgt nun der Aufruf der Methode `download` der Klasse `td.ivaneggel.jsfmanagedbeans.DocumentProvidingController`. Zusätzlich zur URL wird der Speicherort (auf dem Server) für das Dokument als Parameter erwartet (Code-Abschnitt 9-11). Dieser Speicher-Ort in Form einer `java.io.File`-Instanz wird aus dem `ApplicationBean1` mit der Methode `getRelativePathUploadedDocumentsDirectory` plus dem Namen des Dokuments (aus dem Pfad des User-Inputs herleitbar) gebildet.

```
DocumentProvidingController dpc = new
DocumentProvidingController();
dpc.download(destination, path);
```

Code-Abschnitt 9-11 Erzeugen eines DocumentProvidingController-Objekts

9.4 Indizieren eines Verzeichnisses auf dem Server

Wie Sie bereits wissen, erlaubt meine Applikation die Indizierung eines gesamten Server-Verzeichnisses. Für die Bereitstellung ist es nötig, die relevanten Files in ein öffentliches Verzeichnis zu kopieren.

9.4.1 Kopieren der relevanten Dokumente

Für die Indizierung eines gesamten Server-Verzeichnisses benötigt meine Applikation die User-Eingabe des zu indizierenden Verzeichnisses in ein Text-Feld.

Im JSP-Dokument *IndexServerdirectory.jsp* ist daher das `h:inputText`-Tag mit dem Wert des Strings `directoryPath` (Instanzvariable) der Klasse (Managed-Bean) `IndexServerdirectory` verbunden. Der über diesen String erhaltene Wert stellt demnach das zu indizierende Verzeichnis dar. Durch das User-seitige Auslösen des Action-Events per Betätigung des Buttons, wird das Kopieren der Files und die darauf folgende Extraktion und Indizierung eingeleitet (Code-Abschnitt 9-12).

```
<h:inputText
   value="#{Index$IndexServerdirectory.directoryPath}"/>
<h:commandButton
   action="#{Index$IndexServerdirectory.button1_action}"
   value="Find and Index"/>
```

Code-Abschnitt 9-12 Definiton der Text-Feld- und Button-Komponente im JSP-File

Das Managed-Bean `IndexServerdirectory` enthält die Methode `setPathsOfRelevantFiles`, welche ihrerseits innerhalb der Action-Methode `button1_action` aufgerufen wird (Code-Abschnitt 9-13). Wie bereits in Kapitel 3.3.3 dargelegt, sucht diese Methode rekursiv nach relevanten Dokumenten im angegebenen Server-Verzeichnis. Nachdem die benötigten Dateien gefunden wurden, werden diese in den öffentlichen Ordner (mittels der Methode `copyToUploadLocation`) kopiert.

```
File dir = new File(directoryPath);
List<String> filepaths = setPathsOfRelevantFiles(dir);
copyToUploadLocation(filepaths);
```

Code-Abschnitt 9-13 Kopie der Dokumente

9.5 AJAX

9.5.1 Grundlagen von AJAX[73]

Fast immer ist es wünschenswert, dem User während einer sich in Verarbeitung befindenden Transaktion ein gewisses Feedback an ausgeführten Schritten zu geben. Stellt dies bei einer lokalen Applikation überhaupt keine Hürde dar, erfordert es bei einer Web-Applikation deutlich mehr Aufwand.

[73] Dieser Abschnitt bezieht sich auf eigenes Vorwissen

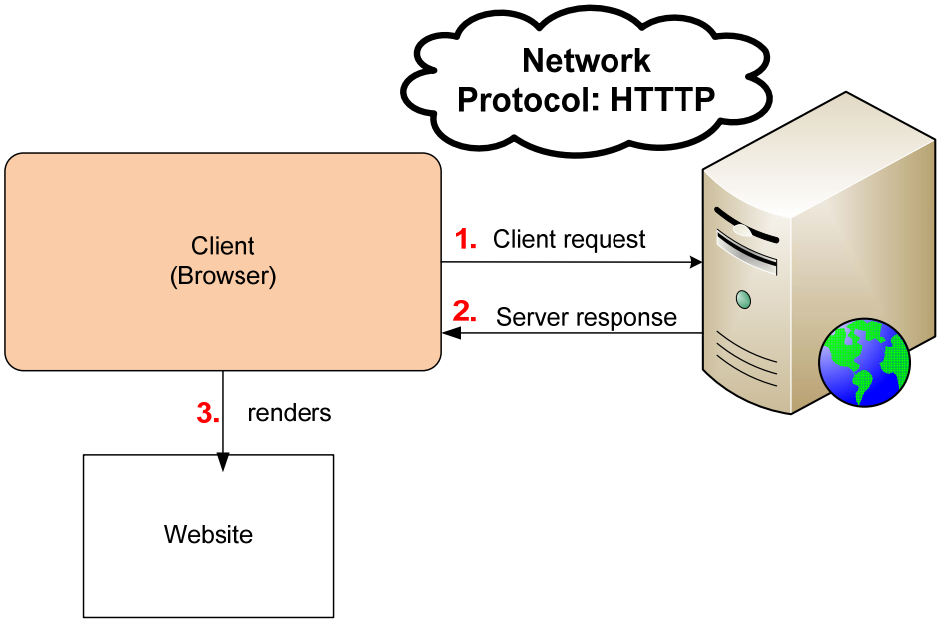


Abbildung 9-1 Kommunikation über HTTP

Der Grund des höheren Aufwands bezogen auf eine Web-Applikation ist im Protokoll http zu suchen. Dieses Protokoll kennt keinen Zustand, d.h. immer wenn der Client (Browser) eine Anfrage an den Server schickt, wird eine neue Verbindung aufgebaut. Dazu muss die entsprechende Web-Seite jedes Mal neu geladen werden (Abbildung 9-1).

Das Bedürfnis, dem User trotz der limitierten Möglichkeiten, Rückmeldung zu geben, nachdem die Seite geladen wurde, kann durch AJAX gelöst werden. Nach dem Client-Request und der darauf folgenden Antwort des Servers ist es die Aufgabe des Browsers, die Web-Seite anzuzeigen. Da in einer Page sowohl HTML (Layout) wie auch Javascript(Client-Funktionalität) erlaubt ist, kann Javascript dazu benutzt werden, ansynchron Anfragen an den Server zu senden (AJAX) und dessen Antwort zu empfangen. Bezogen auf die vom Server erhaltene Response ist JavaScript nun in der Lage, den HTML-Code der Seite zu modifizieren, ohne die Seite neu laden zu müssen (Abbildung 9-2).

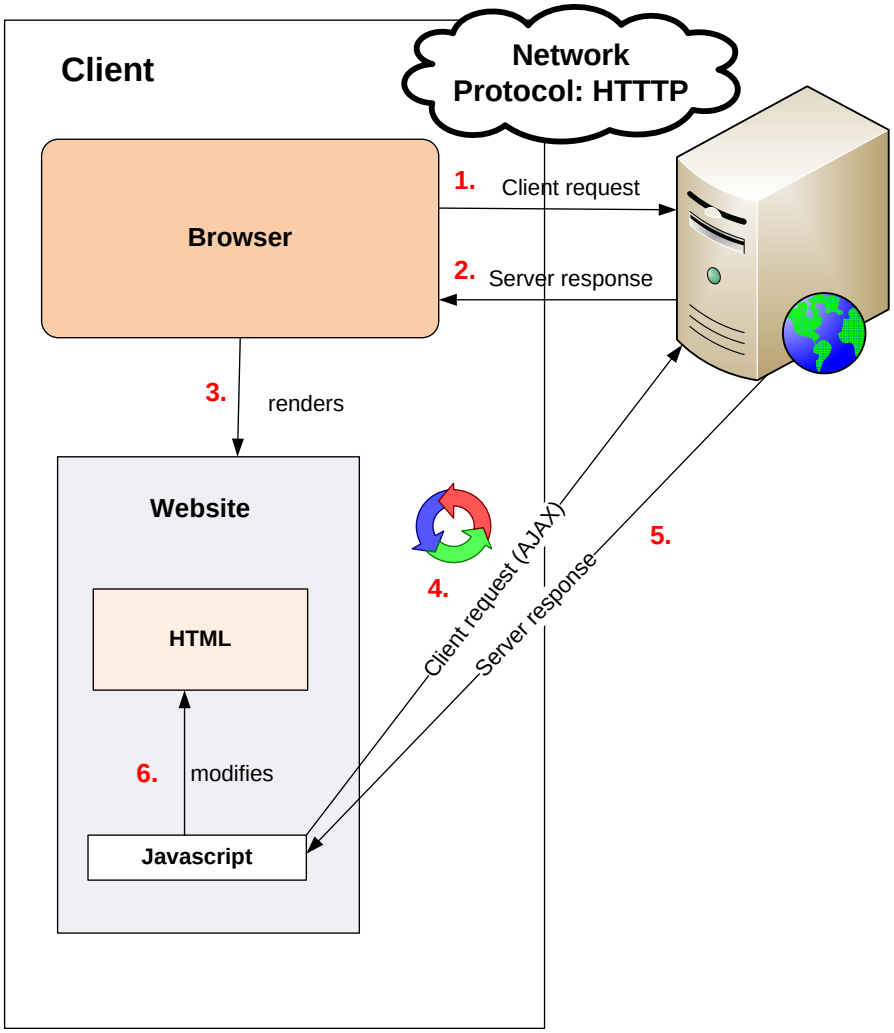


Abbildung 9-2 Ajax-Kommunikation

Im Rahmen meiner Applikation mache ich von dieser Technologie gebrauch, um dem User während des gesamten Indizierungs-Prozesses Rückmeldung zu geben. Wie Sie bereits gesehen haben, wird dieser Prozess vom User bloss als ein einziger Schritt wahrgenommen (ohne AJAX-Funktionalität). In Wirklichkeit wird ein Dokument jedoch zuerst bereitgestellt und Text daraus extrahiert. Damit der User davon Kenntnis nimmt, ist es erforderlich, ihm stetig Rückmeldung zu geben, welcher Prozess ausgeführt wird und welchen Status dieser aufweist.

9.5.2 AJAX-Funktionalität in meiner Applikation

9.5.2.1 Allgemeines

Für die drei verschiedenen Arten der Bereitstellung sind jeweils untereinander differierende Rückmeldung an den User zu geben. Über folgende Schritte der jeweils gewählten Methode soll der User informiert werden:

- **Indizierung eines hochgeladenen Dokuments**
  - Hochladen

- Extraktion
- Indizierung

- **Indizierung eines im WWW erreichbaren Dokuments**
  - Download auf den Server
  - Extraktion
  - Indizierung

- **Indizierung eines Server-Verzeichnisses**
  - Finden relevanter Dokumente
  - Kopieren der Dokumente in ein öffentliches Verzeichnis
  - Extraktion
  - Indizierung

All die oben aufgeführten Schritte sollen folgende Zustände annehmen können:

- *Waiting*
- *Starting*
- *In progress*
- *Finished*

9.5.2.2 Server-Seite

Wird einer der vorhin aufgeführten Schritte gestartet (z.B. Download), muss ein Objekt einer bestimmten Klasse, die nur als reiner Datenhalter fungiert, initialisiert und einer HTTP-Session hinzugefügt werden. Dies geschieht in der Action-Event-Methode (`button1_action`) des jeweiligen Managed-Beans Die Datenhalter-Klasse ist dabei mit jeweils 2 `Boolean`-Attributen versehen, welche angeben ob der Schritt in Gange bzw. abgeschlossen ist. Eine HTTP-Session wird aufgrund der Tatsache gewählt, dass das Objekt zwischen den einzelnen Requests den Zustand behalten muss.

Wird bspw. der Download bei der Indizierung eines im Web erreichbaren Dokuments gestartet, soll ein Objekt der Klasse `DownloadAjax` erzeugt und das Attribut `downloadInProgress` auf `true` gesetzt werden. Nach dieser Massnahme ist es nun erforderlich, eine neue HTTP-Session zu erzeugen, welche in diesem Fall mit dem Keyword `ajaxdownload` verfügbar gemacht wird (Code-Abschnitt 9-14).

```
DownloadAjax d = new DownloadAjax();
d.setDownloadInProgress(true);
this.getSessionMap().put("ajaxdownload", d);
```

Code-Abschnitt 9-14 Hinzufügen eines DownloadAjax-Objekts zu einer Session

Als Server-seitige Kontakt-Möglichkeit für die AJAX-Requests habe ich mich für Java-Servlets entschieden. Aufgabe eines solchen Servlets ist es, sich mit der jeweiligen HTTP-Session zu verbinden, deren Zustand abzufragen, und je nach Status eine Antwort zurückzuliefern. Nachfolgend finden Sie ein Beispiel-Auszug des Servlets, welches für die Abfrage des Download-Status zuständig ist (Code-Abschnitt 9-15).

```
DownloadAjax da = (DownloadAjax)request.getSession().
             getAttribute("ajaxdownload");
if (da==null)
{
    out.println("-2");
    return;
}
if (da.getDownloadInProgress()==false)
{
    out.println("-1");
    return;
}
if (da.getDownloadInProgress() &&
        da.getDownloadFinished()==false
{
    out.println("0");
    return;
}
if (da.getDownloadInProgress()&& da.getDownloadFinished())
{
    request.getSession().removeAttribute
           ("ajaxdownload");

     out.println("1");
     return;
}
```

Code-Abschnitt 9-15 Abfrage des Status im Servlet

Sobald der Download beendet ist, muss in der entsprechenden Managed-Bean Methode der Zustand des Datenhalter-Objekts verändert werden, indem das Attribut `downloadFinished` auf `true` gesetzt wird. Zudem muss natürlich auch die HTTP-Session aktualisiert werden (Code-Abschnitt 9-16).

```
d.setDownloadFinished(true);
this.getSessionMap().put("ajaxdownload", d);
```

Code-Abschnitt 9-16 Update der Session

Aufgrund der auf den Download des Dokumentes folgenden Extraktion muss als nächstes das Datenhalter-Objekt für die Extraktion instanziiert und an die HTTP-Session angebunden werden (Code-Abschnitt 9-17).

```
ExtractionAjax ea = new ExtractionAjax();
ea.setExtractionInProgress(true);
this.getSessionMap().put("ajaxextraction", ea);
```

Code-Abschnitt 9-17 Erzeugung einer Session mit einem ExtractionAjax-Objekt

Ist die Extraktion der Inhalte abgeschlossen, ist es notwendig, im `ExtracionAjax`-Objekt das entsprechende Attribut zu verändern und die HTTP-Session zu aktualisieren (analog zum Download).

Aufgrund auftretender Redundanzen des Codes habe ich mich entschieden, auf die weitere Auflistung desjenigen zu verzichten. Für die anschliessend anstehende Indizierung des Dokuments kann sinngemäss zu den vorigen Schritten vorgegangen werden. Das aufgeführte Beispiel tangiert *nur* den Anwendungsfall der Indizierung eines im WWW erreichbaren Dokuments, jedoch spielt sich diese Prozedur auch bei den beiden anderen Methoden der Bereitstellung ab. Alle von AJAX-kontaktierten Server-Ressourcen und Datenhalter-Klassen befinden sich im Paket `td.ivaneggel.xfeatures.ajax`.

9.5.2.3 Client-Seite

Um den jeweiligen Status mittels AJAX-Request abzufragen, muss das gewünschte Servlet auf dem Server ständig in bestimmten Zeit-Intervallen abgefragt werden. Diese Technik wird als sogenanntes AJAX-Polling bezeichnet. Für diese Aufgabe habe ich die Open-Source-Javascript-Bibliothek Prototype[74] zugezogen. Mit einem sogenannten `PeriodicalExecuter`-Objekt wird der Server nun alle 3 Sekunden abgefragt. Erhält die Javascript-Funktion die Antwort des Servers, wird diese entsprechend verarbeitet und auf der Web-Seite angezeigt.

Wie Sie evtl. bereits aus dem Servlet-Code (Code-Abschnitt 9-15) entnehmen konnten, kann der Server folgende Antworten für einen AJAX-Request anbieten.

- **-2**
  Diese Antwort bedeutet, dass die gewünschte Operation noch nicht ausgeführt wurde (Waiting).
- **-1**
  Anhand dieser Response stellt der Client fest, dass sich die gewünschte Operation in der Start-Phase befindet (Starting).
- **0**
  Die als Antwort erhaltene Ziffer 0 sagt aus, dass die gewünschte Operation in Verarbeitung ist (In progress)
- **1**
  Falls der Client diese Zahl als Response zurückgeliert bekommt, weiss er, dass die gewünschte Operation erfolgreich ausgeführt wurde (Finished).

Es erfolgt nun eine Abfrage der Server-Antwort im Javascript-Code (Code-Abschnitt 9-18). Entsprechend des Wertes, wird das mit der Anzeige des Status beauftragte HTML-Element mit dem erhaltenen Wert in Wort-Form aktualisiert.

[74] Download: http://www.prototypejs.org/assets/2008/1/25/prototype-1.6.0.2.js

```
If(resp=="0")
{
        el.innerHTML = "In progress";
}
```

Code-Abschnitt 9-18 Javascript-Abfrage der Server-Antwort

Damit der Javascript-Code ausgeführt wird, muss dem Submit-Button der Web-Site die entsprechende Funktion (in der Javascript-Datei) als On-Click-Event im dazugehörigen JSP-File angegeben werden (Code-Abschnitt 9-19).

```
<h:commandButton
    action="#{Index$IndexPublicFile.button1_action}"
    id="button1" onclick="showIndexPublicFile()" />
```

Code-Abschnitt 9-19 Definition eines Buttons mit Javascript-Funktionalität im JSP-File

Alle in meiner Applikation benötigten AJAX-Funktionen befinden sich im File *web/Javascript/Ajaxfunctions.js* meines Projektes. Ein vollständiges Beispiel für solche eine Funktion finden sie im Anhang.

9.6 Textuelle-Suche

9.6.1 Zusammensetzung eines Such-Resultates

Die textuelle Suche meiner Applikation beinhaltet die Suche nach Dokumenten anhand des Inhalts und des Autors. Sobald der User einen Suchbegriff ins Text-Feld eingegeben und den Search-Button betätigt hat, werden die gefundenen Resultate (Dokumente) aufgelistet (gemäss Abbildung 9-3).

Ein gefundenes Resultat besteht aus folgenden Komponenten:

- File-Name
- Abstract
- Standort des Dokuments auf dem Server
- Autor
- Link zu den enthaltenen Bildern
- Link zur Dokument-Datei
- Link zu ähnlichen Dokumenten

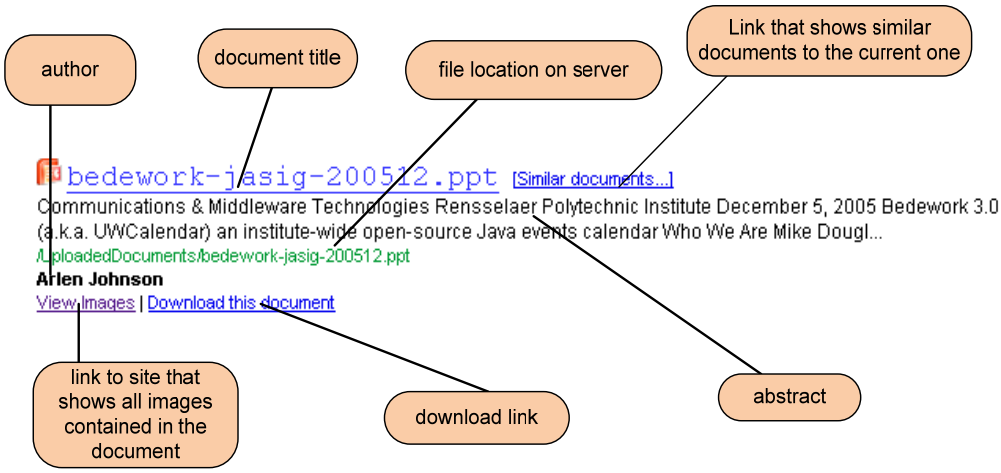


Abbildung 9-3 Aufbau eines Query-Resultates einer textuellen Suche

9.6.2 Art der Suche

Die für die Suche zuständige Managed-Bean-Klasse heisst `TextSearchController` und befindet sich im Paket `td.ivaneggel.jsfmanagedbeans`. Für das Layout ist das File *web/Search/Search.jsp* verantwortlich.

Da der User Dokument-Suchen bezogen auf den Autor wie auch auf den Inhalt durchführen kann, ist im JSP-Dokument ein Drop-Down-Feld (Abbildung 9-4) für die entsprechende Auswahl eingebunden.

```
<h:selectOneMenu
value="#{TextSearchController.selectedItem}">
<f:selectItems value="#{TextSearchController.selectItems}"/>
</h:selectOneMenu>
```

Code-Abschnitt 9-20 Definition der Drop-Down-Komponente im JSP-File

Wie man aus obenstehendem Code entnehmen kann, ist die durch den User getroffene Auswahl im Drop-Down-Feld an ein Attribut namens `selectedItem` der Managed-Bean-Klasse gelinkt.

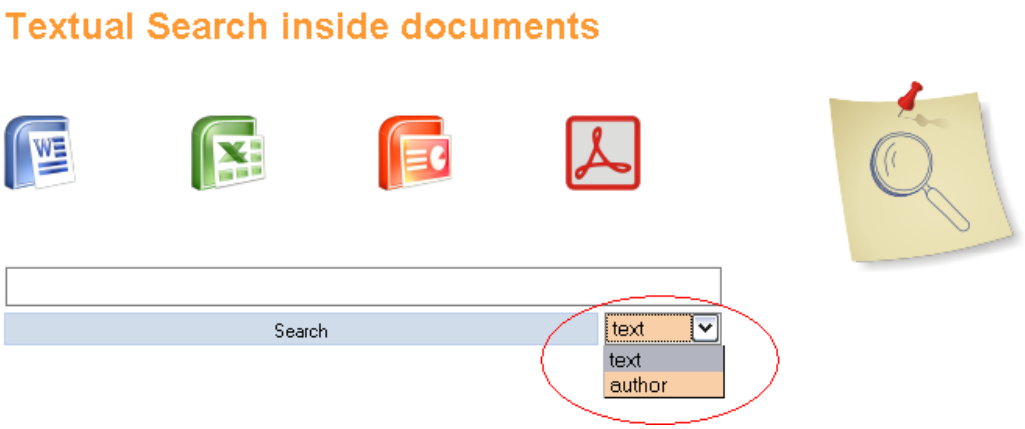


Abbildung 9-4 Drop-Down-Feld zur Auswahl der Such-Art

9.6.3 Ausführen der Suche

Die Such-Resultate werden anhand einer Tabelle dargestellt. Die im Managed-Bean `TextSearchController` enthaltene List-Variable `showListResults` repräsentiert die Daten, welche von der Tabelle angezeigt werden sollen. Durch den Klick des Search-Buttons (Abbildung 9-5) wird diese Variable in der Methode `button1_action` (Code-Abschnitt 9-22) gesetzt. Der Wert für die Variable wird durch die Methode `getSearchResults` (Liste) bestimmt. Als Variable, welche jedes sich in der Liste befindende Objekt repräsentiert, wurde `row` verwendet(Code-Abschnitt 9-21).

```
<h:dataTable value="#{TextSearchController.showListResults}"
var=”row”>
```

Code-Abschnitt 9-21 Definition der Tabelle im JSP-File Search.jsp

```
<h:commandButton action="#{TextSearchController.button1_action
value="Search"/>
```

Code-Abschnitt 9-22 Definition des Buttons im JSP-File Search.jsp

Die Methode `getSearchResults` ihrerseits ruft die Methode getResults auf, die evaluiert, welche Auswahl nun der User (im Drop-Down-Feld) getroffen hat (Code-Abschnitt 9-23).

```
if (selectedItem.equals("text"))
{
     return this.search("content");
}
return this.search("author");
```

Code-Abschnitt 9-23 Aufruf der Methode search mit gewähltem Parameter

Je nach dem im Drop-Down-Feld gewähltem Wert wird die bereits in Kapitel 6 beschriebene Methode `search` aufgerufen, welche das zu durchsuchende Feld als Parameter entgegennimmt.

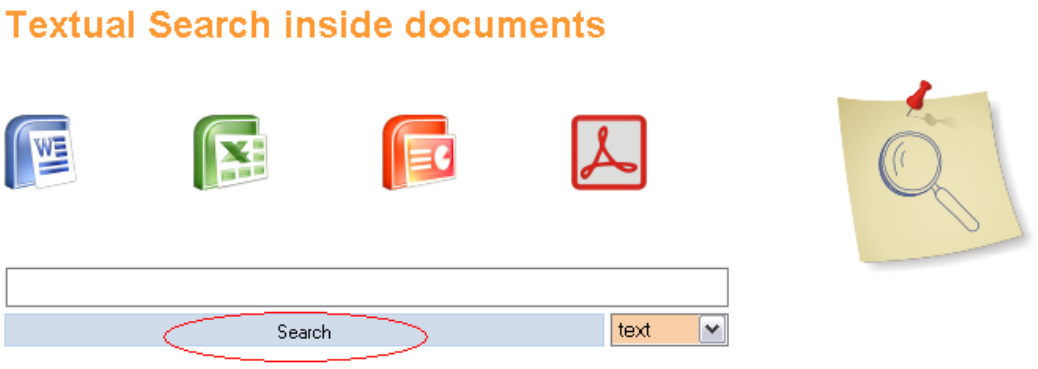


Abbildung 9-5 Suchmaske (Textuelle-Suche)

Sobald die Methode `getSearchResults` nun alle Such-Resultate erhalten hat, werden diese in `SearchResult`-Instanzen (siehe Kapitel 6.3.1.1) gepackt und in Form einer Liste als Return-Value zurückgeliefert.

9.6.4 Anzeige der Resultate

Da die erhaltene Liste mit der Tabelle verknüpft ist, können die Resultate nun angezeigt werden (Abbildung 9-6). Die JSF-Komponente `h:dataTable` (Code-Abschnitt 9-21) iteriert automatisch über die `SearchResult`-Liste und generiert daher die benötigen Zeilen alleine. Aufgabe des Programmierers ist es jedoch, die Werte zu definieren, welche innerhalb einer Zeile angezeigt werden. Ich möchte an dieser Stelle auf die Auflistung zu vieler Details verzichten und beschränke mich deshalb nur auf das Wesentlichste.

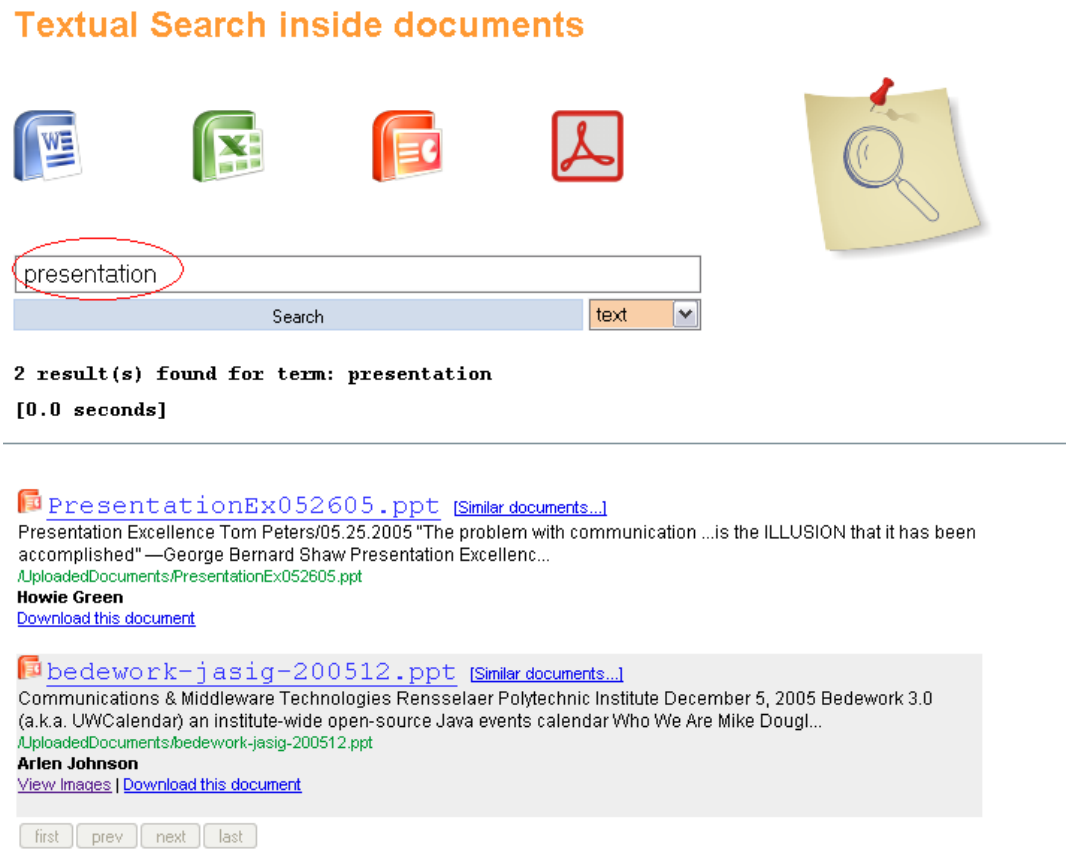


Abbildung 9-6 Anzeige der Reultate einer textuellen Suche

Damit die Informationen der gefundenen Dokumente angezeigt werden können, müssen innerhalb des `h:datatable`-Tags Ausgabe-Komponenten eingebunden werden, welche sich auf ein Attribut eines einzelnen `SearchResult`-Objektes in der an die Tabelle gebundenen Liste des Typs `SearchResult` beziehen. Um bspw. den File-Namen jedes gefundenen Dokuments anzeigen zu lassen, muss dazu folgende Code-Zeile innerhalb der `h:dataTable`-Komponente platziert werden (Code-Abschnitt 9-24).

```
<h:outputText styleClass="documenttitle"
       value="#{row.filename}"/>
```

Code-Abschnitt 9-24 Definition der Output-Komponente im JSP-File

Dieselbe Vorgehensweise gilt auch für das Abstract, den Dokument-Pfad und den Autor. Für den Link zum Dokument (Dokument-Download) wird eine `h:outputLink` Komponente im JSP-File definiert. Diese Komponente entspricht einem gewöhnlichen Hyperlink des HTML-Standards. Der Wert des Links besteht, wie sie aus untenstehendem Code (Code-Abschnitt 9-25) entnehmen können, aus dem Pfad (`publicFilePath`) eines jeden `SearchResult`-Objektes.

```
<h:outputLink
value="#{ApplicationBean1.contextPath}#{row.publicFilePath}">
                <h:outputText value="Download this document"/>
</h:outputLink>
```

Code-Abschnitt 9-25 Definition des Download-Links im JSP-File

Für die Anzeige der im Dokument enthaltenen Bilder muss ebenfalls ein Link gesetzt werden. Dieser Link navigiert zur Seite *PicturePreview.jsp*. Zusätzlich muss der Seite ein GET-Parameter namens *filename* übergeben werden, damit die Seite später weiss, zu welchem Dokument sie die Bilder anzeigen muss. Der Wert dieses Parameters wird aus dem *filename*-Attribut des `SearchResult`-Objektes gewonnen (Code-Abschnitt 9-26).

```
<h:outputLink
value="#{ApplicationBean1.contextPath}/faces/Search/
          PicturePreview.jsp?filename=#{row.filename}">
     <h:outputText value="View Images"/>
</h:outputLink>
```

Code-Abschnitt 9-26 Definition des Links für die Weiterleitung zu PicturePreview inkl. Parameter

9.6.5 Anzeige der im Dokument enthaltenen Bilder

Zur Anzeige der in einem Dokument enthaltenen Bilder (Abbildung 9-7) wurde eigens dafür eine separate Web-Site vorgesehen. Das JSP-File dieser Seite trägt den Namen *`PicturePreview.jsp`* und befindet sich im Verzeichnis *web/Search*. Die Klasse (Managed-Bean) `test4.Search.PicturePreview` ist für die Logik zuständig. Auch für diesen Teil möchte ich nur die relevantesten Schritte erwähnen.

Damit die Seite weiss, für welches Dokument sie die enthaltenen Bilder anzeigen soll, ist es notwendig, wie bereits erwähnt, ihr einen Parameter mit den Namen *filename* zu übergeben.

Abbildung 9-7 Anzeige der in einem Dokument enthaltenen Bilder

Im Managed-Bean PicturePreview befindet sich die im Kapitel 6 beschriebene Methode `displayPictures`. Diese ist zuständig, den Index im Feld *filename* anhand des als Parameter erhaltenen Wertes abzufragen. Dadurch können alle Bilder-Namen (anhand des Fields *image*) bezogen werden. Die erhaltenen Bilder werden im Magaed-Bean dem `List<String>`-Attribut pictures zugeteilt.

Für die Anzeige der Bilder auf der Web-Seite wird wiederum eine Tabelle (`h:dataTable`) verwendet. Der Wert für die Anzeige der Daten in der Tabelle bezieht sich auf das `pictures`-Attribut im Managed-Bean (Code-Abschnitt 9-27).

```
<h:dataTable
value="#{Search$PicturePreview.pictures}" var="row">
```

Code-Abschnitt 9-27 Definition der Tabellen-Komponente mit der row-Varriable, welche jedes einzelne Objekt der Liste repräsentiert.

Um die Thumbnails in der Tabelle anzeigen (Abbildung 9-7) zu können, muss eine `h:graphicImage`-Komponente innerhalb von `h:dataTable` definiert werden, deren Wert sich aus dem Thumbnails-Verzeichnis plus dem Bilder-Namen zusammensetzt. Ebenfalls muss das Bild durch einen Klick auf das Thumbnail-Image in seiner vollen Grösse angezeigt werden. Um dies zu erreichen wird ausserhalb um das Bild herum eine `h:outputLink`-Komponente mit dem Pfad zum Bild platziert (Code-Abschnitt 9-28).

```
<h:outputLink
       value="#{ApplicationBean1.contextPath}/
       #{ApplicationBean1.relativePathExtractedImagesDirectory}/
       #{row[0]}" >
   <h:graphicImage  value="#{ApplicationBean1.
          relativePathThumbnailsDirectory}/#{row[0]}"/>
</h:outputLink>
```

Code-Abschnitt 9-28 Thumbnail-Link zum anzeigen des Original-Bildes

Die zusätzliche Anforderung, durch einen Klick auf den Link unterhalb eines jeden Bildes (Abbildung 9-7) eine GIFT-Suche nach ähnlichen Bildern zu lancieren, verlangt, dass zur Seite *RandomPicture.jsps* navigiert wird. Durch die Übergabe des GET-Parameters *imagename* (Name des Bildes) wird automatisch nach Bildern gesucht, welche dem übergebenen Bild ähneln (Code-Abschnitt 9-29). Dieser Prozess wurde im Kapitel 6.3.2 im Code-Abschnitt 6-9 genauer erläutert.

```
<h:outputLink
value="#{ApplicationBean1.contextPath}/faces/Search/
                RandomPictures.jsp?imagename=#{row[0]}">
        <h:outputText value="similar pictures"/>
</h:outputLink>
```

Code-Abschnitt 9-29 Navigation zur Seite RandomPictures mit Übergabe eines GET-Parameters

9.6.6 Anzeige ähnlicher Dokumente

Ich habe mich dazu entschlossen, dem User die Möglichkeit zu geben, zu jedem als Resultat einer textuellen Suche angezeigten Dokument auf eine einfache und direkte Weise, ähnliche Dokumente zu finden. Die ursprüngliche Idee dabei war, dem `QueryParser` den gesamten Text eines gefundenen Dokuments als Query-String zu übergeben. Dies führte jedoch zu erheblichen Problemen, weil der `QueryParser` aufgrund der eigenen Query-Sprache gewisse Wörter (z.B. *and* oder *or*) als Suchkriterium interpretiert. Bei längeren Dokumenten wurde dabei sogar das Limit der Anzahl solcher Ausdrücke erreicht. Der andere Nebeneffekt war, dass die Suche natürlich nicht korrekt ausgeführt wurde, da die von der Query-Language benutzten Wörter nicht als zu suchender Text, sondern als spezielle Terme in die Suche aufgenommen wurde.

Nach diesem kleinen Rückschlag habe ich beschlossen, nur das Abstract dazu zu benutzen, ähnliche Dokumente zu finden, da dies mit grosser Wahrscheinlichkeit nicht viele geschützte Wörter enthält und ich aufgrund mangelnder zeitlicher Ressourcen nicht mehr in der Lage war, durch Recherche eine bessere Lösung zu finden. Doch auch diese Implementierung bietet nicht eine zufriedenstellende Lösung, daher habe ich es nicht für nötig gehalten, diese an dieser Stelle genauer zu dokumentieren. In einer produktiven Umgebung müsste diese Lösung dringend überdacht und sorgfältiger implementiert werden.

9.7 Visuelle Suche

Die in meiner Applikation verfügbare visuelle Suche mit Bildern soll natürlich ebenfalls über ein Web-Interface ablaufen. Im Kapitel 8.1.2.2 haben Sie bereits erfahren, wie die Ähnlichkeits-Suche und das Empfangen zufälliger Bilder vonstatten geht. Diese Funktionalität muss nun in eine Web-Seite eingebunden werden. Alle mit der visuellen Suche verbundenen Aktionen erfolgen über die Seite *RandomPictures.jsp*. In diesem Abschnitt wird bewusst auf eine Übermässige Auflistung von Code verzichten, da dies deutlich den Rahmen dieser Dokumentation sprengen würde. Ich habe mich hauptsächlich darauf konzentriert, die Funktion etwas genauer zu erläutern.

9.7.1 Empfangen zufälliger Bilder

Das Empfangen zufälliger Bilder und die Anzeige dieser geschieht über die Methode `initPictures` im Managed-Bean `test3.Search.RandomPictures`. Das korrespondierende JSP-File findet sich unter */web/Search/RandomPictures.jsp*.

```
Giftsearcher gf = new Giftsearcher();
gf.connect();
QueryObject[] randomQueryObjects =
        gf.getRandomImages(getNumberImages());
gf.close();

List<OwnQueryObject> list4 = new ArrayList<OwnQueryObject>();
//etc...
```

Code-Abschnitt 9-30 Empfang zufälliger Bilder

Innerhalb der Methode wird die bereits bekannte Methode `getRandomImages` der Klasse `Giftsearcher` aufgerufen (Code-Abschnitt 9-30). Sobald die `QueryObject`-Array-Instanz initialisiert wurde, wird dieses Array in eine Liste des Typs `OwnQueryObejct` (Paket `td.ivaneggel.searching.gift`) gespeichert (wurde im Code-Beispiel nicht extra aufgeführt). Die eben erwähnte Klasse wurde von mir konzipiert da die `QueryObject`-Klasse nicht alle Anforderungen erfüllt, um die für die Anzeige auf der Web-Seite geforderten Informationen von dort aus direkt aufzurufen. Die von mir getätigte Haupt-Erweiterung bezieht such auf das Attribut queried, welche aussagt, ob ein bestimmtes Bild schon einmal abgefragt wurde.

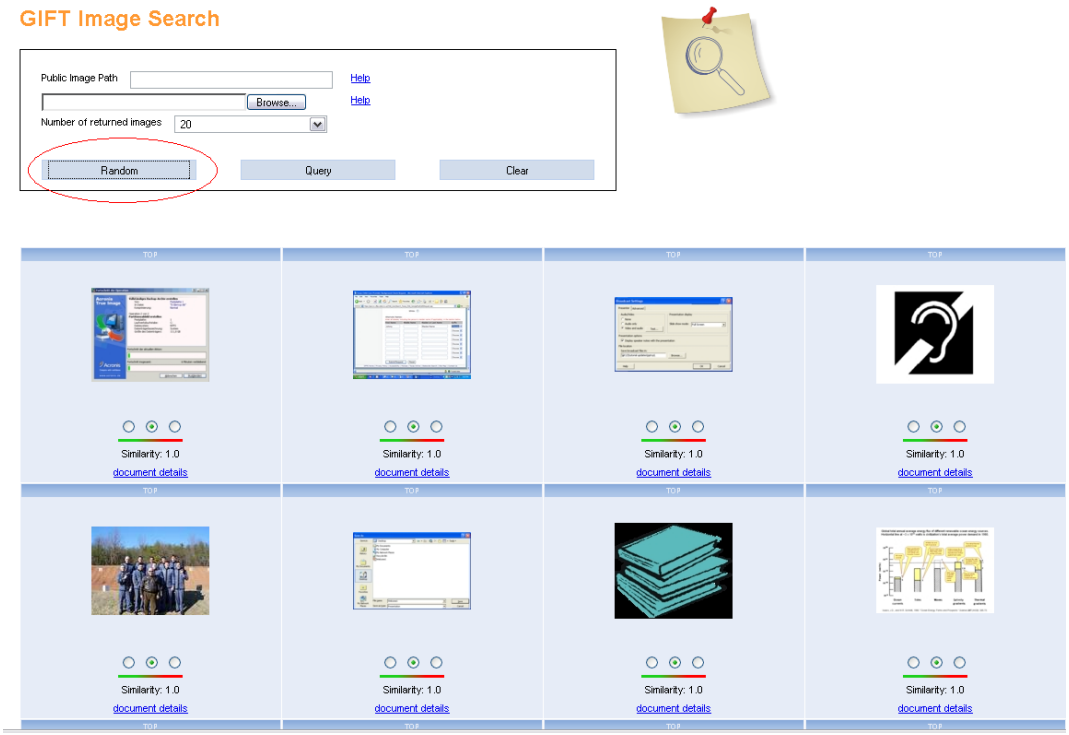


Abbildung 9-8 Empfangen zufälliger Bilder des GIFT-Servers

Durch Klicken des Random-Buttons wird die Methode `initPictures` aufgerufen. Im JSP File ist die Tabelle an das Listen-Attribut pictures gebunden, welches verantwortlich für die Anzeige der Bilder ist.

9.7.2 Ähnlichkeits-Suche

Sobald Bilder auf der Such-Seite angezeigt werden, können diese in die Ähnlichkeits-Suche aufgenommen werden.

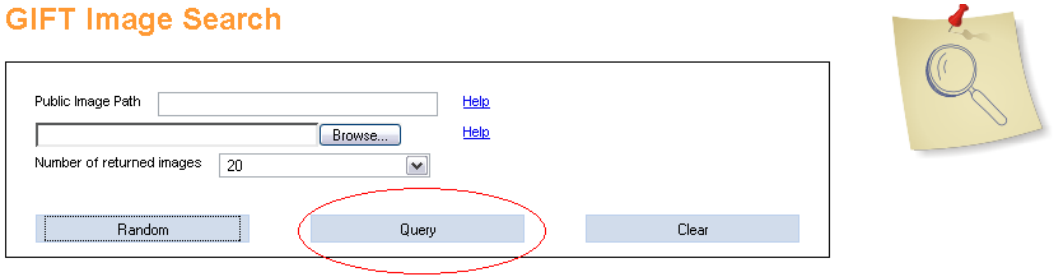


Abbildung 9-9 Query-Button auf der Seite RandomPictures.jsp

Dabei ist der User im Stande, eine Query-By-Example durchzuführen. Bei dieser Art von Query wird dem abzufragenden Bild jeweils eine Relevanz mitgegeben. Für abzufragende Bilder habe ich mich, wie bereits angetönt, für 2 Arten von Relevanzen entschieden: relevant und irrelevant.

Abbildung 9-10 Query By Example

Wie Sie der Abbildung 9-10 Query By Example entnehmen können, kann der User durch den Klick einer der drei sich unter jedem Bild befindenden Radio-Buttons, die Relevanz festlegen. Das erste Bild wird hierbei als relevant, das zweite als irrelevant, das dritte ebenfalls als relevant und das vierte als neutral betrachtet. Nach der Betätigung des Query-Buttons (Abbildung 9-9) werden alle mit einer Relevanz gekennzeichneten Bilder in die Ähnlichkeits-Suche aufgenommen. Ist ein Bild mit dem Relevanz-Faktor neutral (Standard-Fall) markiert, wird es nicht in die Query aufgenommen. Nach der Durchführung der Query werden die Ergebnis-Bilder angezeigt. Die bereits abgefragten Bilder werden hierbei mit dem Vermerk Query Image versehen. Wie vorhin angesprochen wird im `OwnQueryObject`-Objekt das Boolean-Attribut `queried` abgerufen, um festzustellen ob das Bild schon einmal abgefragt wurde. Anhand des Such-Resultates der Bilder kann eine weiterführende Query ausgeführt werden, wobei die bereits abgefragten Bilder mit den neu dazugekommenen für die erneute Abfrage kumuliert werden. Ebenfalls die Betätigung des Random-Buttons verursacht nicht die Nicht-Berücksichtigung der bereits abgefragten Query-Bilder. Um eine komplett neue Suche zu lancieren, ist es notwendig, den Clear-Button zu klicken.

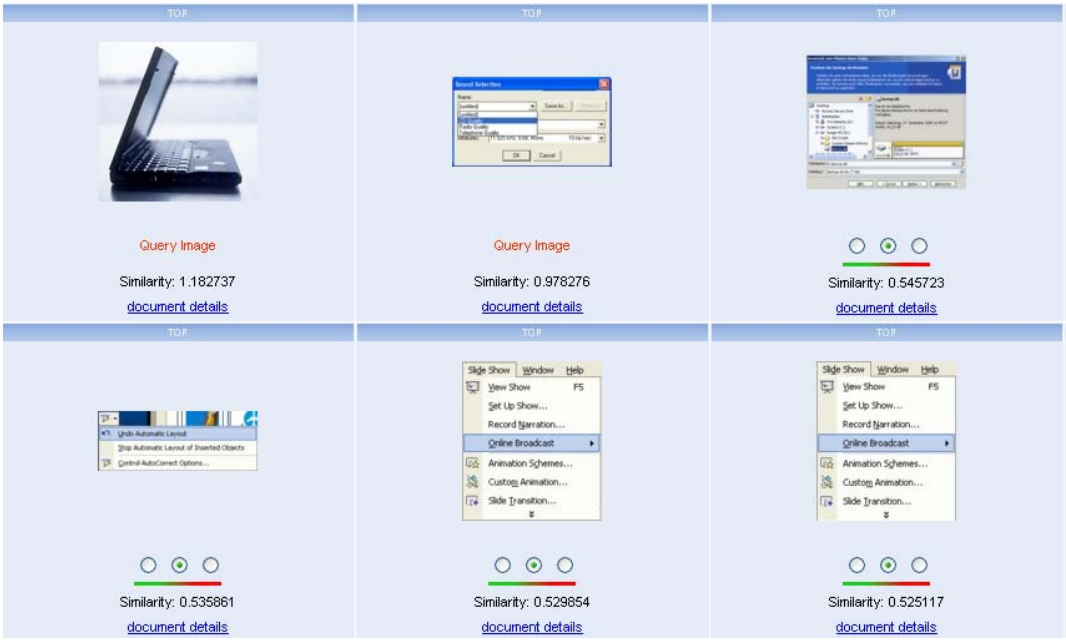


Abbildung 9-11 Query-Resultat

Für die Durchführung der Ähnlichkeits-Suche ist hauptsächlich die Methode `doSearch` der Klasse `RandomPictures` verantwortlich. Ihr Aufgabe ist es, die Klasse `GiftSearcher` zu instannziieren und die Methode `doSimilaritySearch` aufzurufen (siehe Code-Abschnitt 9-31 Methode doSearch).

```
private QueryObject[] doSearch(QueryObject[] qo, int maxResults)
{
   //ACTUAL GIFT QUERY
   Giftsearcher gf = new Giftsearcher();
   gf.connect();
   QueryObject[] result = gf.doSimilaritySearch(qo, maxResults);
   gf.close();
   return result;
}
```

Code-Abschnitt 9-31 Methode doSearch

9.7.2.1 Ähnlichkeits-Suche anhand zufällig empfangener Bilder

Durch das Drücken des Random-Buttons (Abbildung 9-12) auf der Seite *RandomPictures.jsp* werden zufällige Bilder (welche auf dem GIFT-Server indiziert sind) angezeigt. Anhand der empfangenen Bilder kann nun eine Ähnlichkeits-Suche gestartet werden. Ebenfalls unmittelbar nachdem eine der in den nachfolgenden Abschnitten beschriebenen Arten der Ähnlichkeits-Suche ausgeführt wurde, kann der Random-Button betätigt werden. Durch diese Aktion werden wiederum neue zufällige Bilder empfangen, die jedoch bereits abgefragten Bilder werden für eine erneute Abfrage wieder berücksichtigt.

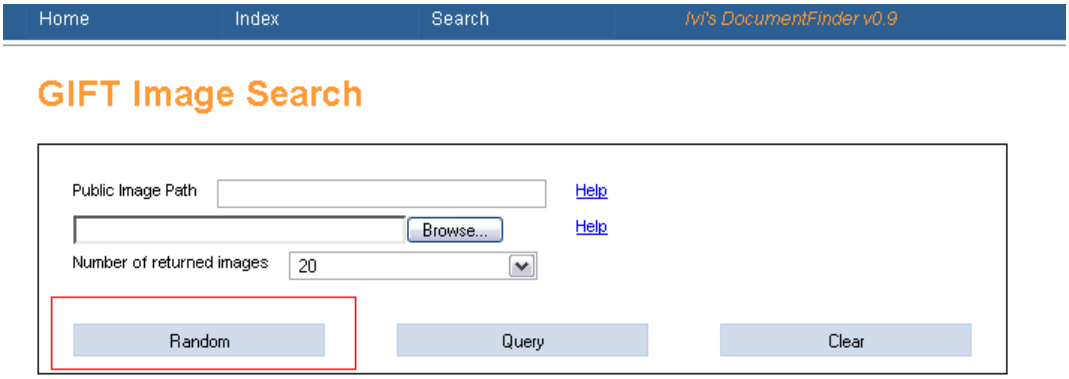


Abbildung 9-12 Random-Button auf der Seite RandomPictures.jsp

9.7.2.2 Ähnlichkeits-Suche anhand eines Bildes, das in einem durch die textuelle Suche gefundenen Dokument enthalten ist

Wie bereits beschrieben ist die Seite *RandomPictures.jsp* fähig, den Parameter *imagename* zu erhalten, und anhand dessen (Name des Bildes) eine Ähnlichkeits-Suche durchzuführen (mit einer Relevanz von *1*). Die gefundenen ähnlichen Bilder werden daraufhin auf derselben Seite angezeigt. Anhand der gefundenen Bilder kann die Ähnlichkeits-Suche fortgeführt werden. Durch die Betätigung des Clear-Buttons kann eine neue Suche gestartet werden (die vorher abgefragten Bilder werden nicht mehr berücksichtigt).

9.7.2.3 Ähnlichkeits-Suche anhand eines hochgeladenen Bildes

Mit Gift es ist möglich, auch externe Bilder in die Query aufzunehmen. Möchte der User nun ein auf seiner Festplatte gespeichertes Bild in die Abfrage miteinbeziehen, kann er dies über die HTML-Komponente Input-File (Abbildung 9-13 Hochladen eines Bildes für die Query) abwickeln.

Abbildung 9-13 Hochladen eines Bildes für die Query

Das Bild wird auf die Maschine des Applikations-Servers (nicht auf den GIFT-Server) hochgeladen. Dies geschieht über die Methode `upload` der Klasse `RandomPictures`. Das Verzeichnis *UploadedQueryImages* (definiert in der

Datei *App_Properties.properties*) fungiert dabei als Container für die hochgeladenen Bilder. Zum Bild, welches hochgeladen wird, wird durch die Methode `makeThumbnailForUploadedPicture` zusätzlich ein Thumbnail generiert und in das Verzeichnis *UploadedQueryImagesthumbs* geschrieben.

Nach dem erfolgreichen Hochladen des Bildes wird dieses in die Query aufgenommen (als relevant). Dabei ist es essentiell, dass der Applikations-Server öffentlich erreichbar ist. Ist dies nicht der Fall, ist es dem GIFT-Server unmöglich, das abzufragende Bild in die Query mit-einzubeziehen.

Das untenstehende Beispiel soll verdeutlichen, wie in etwa die Ähnlichkeits-Suche über ein hochgeladenes Bild erfolgt. Im Prinzip läuft alles genau gleich ab wie mit einem Bild, dass dich auf dem GIFT-Server befindet. Der einzige Unterschied ist, dass das das location Attribut des `QueryObjects` sich nicht auf die Adresse des GIFT-Servers bezieht (Code-Abschnitt 9-32).

```
QueryObject[] qo = new QueryObject[1];
qo[0] = new QueryObject();
qo.location = new
        URL(appserveraddress+uploadedImageDirectory+
        "/"+picname);
doSearch(qo, maxResults)
```

Code-Abschnitt 9-32 Query-Beispiel eines hochgeladenen Bildes

Sofern nicht der Clear-Button betätigt, wird das hochgeladene Bild in jede folgende Query mit-einbezogen.

9.7.2.4 Ähnlichkeits-Suche anhand eines im WWW verfügbaren Bildes

Wie bereits im vorhergehenden Abschnitt (9.7.2.1) beschrieben, können externe Bilder in die GIFT-Abfrage aufgenommen werden (Abbildung 9-14). Anhand der Angabe des Pfades, kann der User ein im WWW verfügbares Bild in die Abfrage mit-einbeziehen. Dabei wird der String des vom User eingegebenen Pfades dem `QueryObject` hinzugefügt. Die Abfrage geschieht auf übliche Weise und wird daher nicht noch einmal aufgeführt. Wie auch bei sonstigen Queries, wird das Bild bei jeder nachfolgenden `Query` nochmals abgefragt, sofern nicht der Clear-Button betätigt wurde.

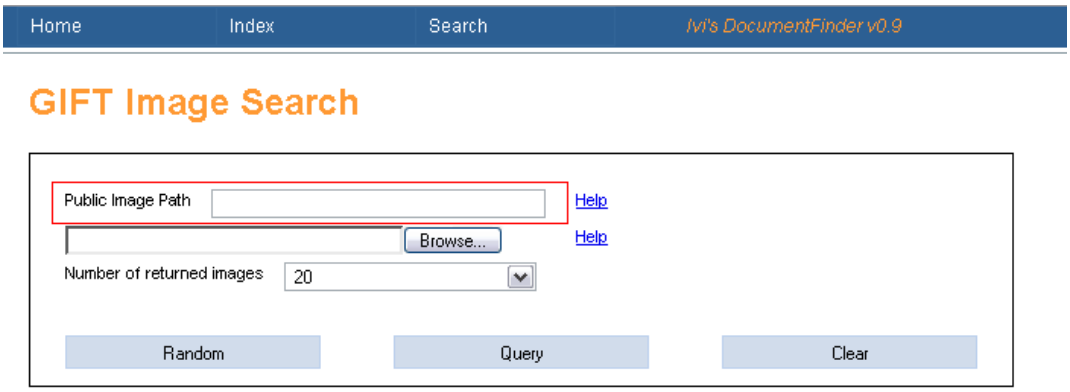


Abbildung 9-14 Angeben eines öffentlichen Pfades

9.7.2.5 Document details

Wie schon im Kapitel 6.3.2 angetönt, ist es möglich, anhand eines Bildes wieder das dazugehörige Dokument zu finden. Dies geschieht durch den Klick auf den Link *Document Details* auf der Seite *RandomPictures.jsp* (Abbildung 9-15).

Abbildung 9-15 Link zur Seite DocumentDetails.jsp

Es ist lediglich notwendig, den Bilder-Namen der Seite *DocumentDetails.jsp* als Parameter *image* mitzugeben (Code-Abschnitt 9-33). Gemäss Kapitel 6.3.2 ist die Methode `initFields` des Managed-Beans `DocumentDetails` für die Lucene Abfrage und die anschliessende Anzeige der Dokument-Details auf der Web-Seite zuständig (Abbildung 9-16).

```
<h:outputLink rendered="#{row[1]!=null}"
   value="#{ApplicationBean1.contextPath}/
   faces/Search/DocumentDetails.jsp?
   image=#{row[0].filename}">
  <h:outputText value="document details"/>
</h:outputLink>
```

Code-Abschnitt 9-33 Link zur Navigation zu DocumentDetalis.jsp mit Parameter image

Abbildung 9-16 Anzeige der Dokument-Details

10 Konklusion / Zukunft

Die von mir entwickelte Applikation bietet dem End-User ein Tool für die Indizierung und die Suche von Dokumenten aufgrund textueller und visueller Kriterien. Im Rahmen der Entwicklung meiner Lösung habe ich eine Reihe an interessanten Tools kennengelernt.

Die Bibliothek Lucene für die Indizierung bzw. Suche von Text ist mir dabei besonders positiv aufgefallen. Trotz der erheblichen Komplexität der Materie bezüglich Indizierung und Suche versteht es Lucene, den Anwender nicht zu überfordern und gewährt ihm damit eine schnelle und stabile Lösung mit Hilfe derer die gewünschten Funktionalitäten in kürzester Zeit implementiert werden können. Die textuelle Suche mit Lucene ist extrem Leistungs-stark. Oft betrug die von mir vorgenommene Zeitmessung bezüglich der Suche weniger als eine Millisekunde bei ca. 100 indizierten Dokumenten.

Im Rahmen der Extraktion relevanter Inhalte der Dokument-Typen Word, Excel und Powerpoint war die Arbeit mit POI sehr intuitiv, sodass es nur selten von Nöten war, in der Dokumentation nachschlagen zu müssen. Auch das Ergebnis ist der Extraktion von Text und Bild war sehr zufriedenstellend. Ein kaum nennenswerter Nachteil ist, dass die verschiedenen APIs innerhalb von POI untereinander nicht denselben Aufbau aufweisen, was ab und zu ein wenig für Verwirrung sorgte. Nichtsdestotrotz ist dieses API höchst empfehlenswert.

Das für die Extraktion der PDF-Dokumente verwendete PDFBox-API war im Allgemeinen sehr einfach anzuwenden. Bei den Ergebnissen gab es jedoch einiges zu bemängeln. Beispielsweise wurden die Bilder immer spiegelverkehrt extrahiert und es wurde teilweise Text als Bild erkannt. Bei der professionellen Anwendung meiner Applikation müsste dies dringend berücksichtigt und ggf. ein besseres API verwendet werden.

Wie bereits erwähnt, wurde die Indizierung der Bilder mit GIFT manuell vorgenommen. In einem nächsten Schritt wäre es natürlich sinnvoll (notwendig), diese Indizierung automatisch (gemeinsam mit der Indizierung des Textes) ablaufen zu lassen. Idealerweise sollte daher von meiner Applikation aus ein GIFT-Web-Service aufgerufen werden können, der sich um diese Aufgabe kümmert.

Bezüglich GIFT und der damit verbundenen Such-Funktion war ich positiv überrascht, wie gut das Tool mit Hilfe des vom User mitgegebenen Relevanz-Feedbacks seinen Dienst erledigt. Für die visuelle Suche mit GIFT wurde SnakeCharmer als Client benutzt. Obwohl dieses Programm (in meinem Fall nur als Bibliothek verwendet) sehr gute Resultate lieferte, sollte es zu einem späteren Zeitpunkt ersetzt werden, da doch schon eine nicht unerheblich Anzahl der darin enthaltenen Klassen und Methoden veraltet sind bzw. nicht mehr dem Java-Standard entsprechen.

Was die Entwicklung meiner Applikation anbelangt ist zu erwähnen, dass alles relativ rasch und ohne grössere Komplikationen verlief. Im Grossen und Ganzen bin ich mit meiner Arbeit sehr zufrieden. Dessen ungeachtet könnten natürlich

noch einige Verbesserungen vorgenommen werden. Bspw. muss die textuelle Suche nach ähnlichen Dokumenten überdacht werden, da sie von mir nur unsorgfältig implementiert wurde. Ein weiterer Schritt könnte das Erstellen eines Bildes (im Extraktions-Prozess) anhand jeder in einem Dokument enthaltenen Seite sein. Somit könnten alle Seiten eines jeden Dokuments mit GIFT indiziert werden. Dies wäre bspw. sehr vorteilhaft für die Suche nach PowerPoint-Folien.

Da nur knappe zeitliche Ressourcen für die Realisierung meiner Bachelor-Arbeit zur Verfügung standen und ich daher nicht alle gewünschten Features einbauen konnte, hoffe ich doch, dass die von mir hervorgebrachte Applikation als Grundgerüst für weitere Entwicklungen in dieser Richtung dienen kann.

Zu guter Letzt möchte ich erwähnen, dass durch die hervorragende Betreuung durch Dr. Henning Müller, mit seinen klaren Vorstellungen und konkreten Zielvorgaben, ein produktives Arbeiten von Anfang an möglich war.

11 Ehrenwörtliche Erklärung

Ich bestätige hiermit, dass ich die vorliegende Diplomarbeit alleine und nur mit den angegebenen Hilfsmitteln realisiert habe und dass ich ausschliesslich die erwähnten Quellen benutzt habe. Ohne Einverständnis des Leiters des Studienganges und des für die Diplomarbeit verantwortlichen Dozenten Hr. Dr. Henning Müller werde ich dieses Dokument an niemanden verteilen, ausser an die Personen, welche mir die wichtigsten Informationen für die Verfassung dieser Arbeit geliefert haben.

Sierre, November 2008

12 Quellen-Verzeichnis

12.1 Gedruckte Quellen

[Gospodnetic, Hatcher (2004)]

Otis Gospodnetic, Erik Hatcher, Lucene in Action, 2005, Mamming Publications, ISBN 1-932394-28-1

[Squire, Müller, Müller (1999)]

David McG. Squire, Wolfgang Müller, Henning Müller, Relevance feedback and term weighting schemes for content-based image retrieval, In Third International Conference On Visual Information Systems, *Springer Lectures Notes in Computer Science LNCS 1644*, pages 549-556, Amsterdam, The Netherlands, 2-4 June 1999

12.2 Online-Quellen

Die Online Quellen mit dem jeweiligen Stand wurden innerhalb des Dokuments in den Fussnoten bereits angegeben. Daher wird an dieser Stelle verzichtet, diese nochmals aufzuführen.

13 Abkürzungsverzeichnis

AJAX	Asynchronous Javascript And XML
API	Application Programming Interface
CBIR	Content Based Image Retrieval
DIE	Integrated Development Environment
DMZ	Demilitarized Zone
GIFT	GNU Image Finding Tool
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
IR	Information Retrieval
JSF	JavaServer Faces
JSP	JavaServer Pages
JVM	Java Virtual Machine
MRML	Multimedia Retrieval Markup Language
MVC	Model-View-Controller
PDF	Portable Document Format
QBE	Query By Example
RAD	Rapid Application Development
URL	Uniform Resource Locator
WWW	World Wide Web
XML	Extensible Markup Language

14 Anhang

14.1 Beispiel einer AJAX-Funktion

```
function showIndexPublicFile()
{
    showDownloadPercentage();
}
```

```
function showDownloadPercentage()
{    var finished = false;
    var executer = new PeriodicalExecuter(
    function(){
        new Ajax.Request('../DownloadServlet', {
            method:'post',
            onComplete: function(transport) {
               var resp = transport.responseText;
               var el =
        document.getElementById
       ("indexform:outputTextDownloadPercentage");

               resp = resp.replace(/^\s+|\s+$/g, ''); //trimming
               if(resp=="-2")
               {
                   if (finished)
                   {
                        executer.stop();
                   }
                   else
                   {
                        el.innerHTML = "Waiting...";
                   }
               }
               else if(resp=="-1")
               {
                    el.innerHTML = "Starting...";
               }
               else if(resp=="0")
               {
                    el.innerHTML = "In progress";
               }
               else  if (resp=="1")
               {
                   executer.stop();
                   finished = true;
                    el.innerHTML = "Download Finished";
                    showExtractionPercentage();
               }

                }
            });}, 3); //INTERVALL (IN SECONDS)
    executer.execute();}
```

14.2 Konfigurations-File App_Properties.properties

```
#Location of Lucene-index on Server filesystem
index-location = c:\\temp\\testindex

#Location of extracted Images (Should be a relative Path of an public accessible
#directory in the Webapplication context)
images-location = /ExtractedImages

#Location of created Thumbnails of extracted Images (Should be a relative Path of
a public accessible
#directory in the Webapplication context)
thumbnail-location = /Thumbnails

#Location of uploaded Documents (Should be a relative Path of an public
accessible
#directory in the Webapplication context)
upload-location = /UploadedDocuments

#Minimal width of an image contained in a document (in pixels) that should be
saved and indexed
max-image-width = 32
#Minimal height of an image contained in a document (in pixels) that should be
saved and indexed
max-image-height = 32

#Generated thumbnail size : Expresses the maximal height and width of a thumbnail
in pixels
#This value should not be greater than 150 because the table displaying the image
is set for a
#maximal height and width of 150px
generated-thumbnail-maximal-size = 110

#Public Server IP
public-server-ip = 153.109.124.56
#Application Server Port (Glassfish)
application-server-port = 8080

#SETTINGS FOR GIFT/Image Search ----------------------------------------------
------------
#GIFT Server
gift-server-ip = 129.194.97.226
#GIFT Port
gift-server-port = 12755
#User
gift-server-user = anonymous
#Algorithm-to-use
gift-default-algorithm-id = adefault
#Collection to use
gift-default-collection = c-37-50-13-30-8-108-2-273-0
#default public directory for pictures on the gift server
gift-default-images-directory = /images/hesso
#location of picture that is used as query picture thumbnail
#when user has chosen a public picture to query
public-thumbnail-location = /resources/pics/publicpicture.jpg
#location of uploaded query-images
uploaded-query-images-location = /UploadedQueryImages
#location of uploaded query-images thumbs
uploaded-query-images-thumbs-location = /UploadedQueryImagesThumbs
```

14.3 Pflichtenheft

PROVISORISCHES PFLICHTENHEFT

10.09.2008

BACHELOR-ARBEIT HES-SO VS, WIRTSCHAFTSINFORMATIK

Ivan Eggel
Sandstrasse 26
3904 Naters

Betreuer: Dr.Henning Müller

0 Einführung

Dieses Dokument soll die Anforderungen an das Endprodukt meiner Bachelorarbeit beschreiben. Da 8 Wochen ein sehr kurzer und viel zu knapper Zeitraum für so ein Projekt sind, um alle Anforderungen zu erfüllen, sind diese nachfolgend unterteilt in 2 Kriterien:

- **Sollkriterien**: Erfüllung dieser Kriterien wird angestrebt
- **Kannkriterien**: Erfüllung dieser Kriterien ist nicht notwendig, werden nur angestrebt falls überschüssige Kapazitäten vorhanden sind.

Mein Projekt kann grob in 4 Arbeitsschritte aufgeteilt werden. Diese sind:

1. **Extraktion** des Inhalts komplexer Dateien
2. **Automatisierung** der Extraktion
3. **Indexierung** des Dokuments
4. Webinterface für die **Durchsuchung** des Index

Als nächstes wird nun jeder dieser Arbeitsschritte in verschiedene Anforderungen unterteilt, welche dann jeweils den Soll- oder Kannkriterien zugeordnet werden.

1. Extraktion des Inhalts komplexer Dateien

1.1 Dateiformate

Ich habe mich für vier unterstützte Dateiformate entschieden. Diese sind:

1. doc (MS Word 97 – 2003) (**Sollkriterium**)
2. xls (MS Excel 97 – 2003) (**Sollkriterium**)
3. ppt (MS Powerpoint 97 – 2003) (**Sollkriterium**)
4. PDF (**Kannkriterium**)

Das Endprodukt soll dem User also ermöglichen, von diesen Dateiformaten Inhalte extrahieren zu lassen, um diese für die nachfolgende Indexierung bereitzustellen (wobei Extrahierung und Indizierung für den User nur einen einzigen erkennbaren Arbeitsschritt darstellen können).

1.2 Extraktion von Text (Sollkriterium)

Es soll möglich sein, den gesamten Text der jeweils oben genannten Dokumentformate zu extrahieren. Der gesamte Text wird indiziert, jedoch nicht als Text abgespeichert. Dies soll eine Volltextsuche ermöglichen, ohne jedoch unnötig Unmengen an Text auf der Festplatte abzuspeichern. Es ist nicht ausgeschlossen, dass Sonderzeichen falsch extrahiert werden. Dies hängt vom jeweiligen API ab, das verwendet wird.

1.3 Extraktion von Meta-Informationen

Um dem User zusätzliche Informationen bei einem Suchergebnis anzuzeigen ist es wünschenswert, zusätzlich zum Text des Dokuments auch Meta-Daten zu extrahieren.

1.3.1 Extraktion Abstract (Sollkriterium)

Es ist sinnvoll, ein kurzes Abstract des extrahierten Textes abzuspeichern, um dem User, wenn er nach Dokumenten sucht, einen kurzen Überblick zu gewähren, um was sich das Dokument handeln könnte. Dieses Abstract wird dann jeweils nur gespeichert und nicht indexiert. Voraussichtlich stellt das Abstract die ersten 100 Zeichen des Dokuments dar.

1.3.2 Extraktion Autor (Sollkriterium)

Sofern vorhanden und möglich, wird zu jeder Datei zusätzlich noch der Name des Autors extrahiert und indexiert. Dies soll einer eventuell späteren Implementierung einer Suche nach Autoren dienen. Zudem wird zu jedem Suchergebnis der Autor (sofern vorhanden) beigefügt.

1.4 Extraktion von Bildern

Ebenfalls soll eine Extraktion von Bildern innerhalb eines Dokuments erfolgen. Hierbei wird der Fokus vor allem auf die im WWW gängigen Bildformate JPG, GIF und PNG gelegt. Diese Dateien werden mit einem eindeutigen Namen in einem öffentlich zugänglichen Ordner abgelegt, um den Zugriff auf diese (von einem User aus dem Web) zu gewährleisten.

1.4.1 Extraktion aller kompatiblen Bilder (Sollkriterium)

Alle enthaltenen Bilder (sofern kompatibel) des Dokuments werden extrahiert und anschliessend abgespeichert.

1.4.2 Generierung von Thumbnails (Sollkriterium)

Um dem User einen kurzen Überblick über die im Dokument enthaltenen Bilder zu geben, wäre es sinnvoll, Thumbnails (verkleinerte Bilder) der jeweils extrahierten Bilder in der selben Prozedur auch gleich zu generieren und simultan zu den Originalbildern abzuspeichern.

1.4.3 Berücksichtigung irrelevanter Bilder (Kannkriterium)

Es ist durchaus üblich, dass ein Dokument irrelevante Bilder wie z.B. Icons oder Symbole enthält. Diese sind normalerweise kleiner (in Pixeln) als die relevanten Bilder. Falls das Programm solche findet, sollen diese nicht abgespeichert werden. Bilder gelten als irrelevant, falls die Dimension derer kleiner als 16x16 Pixel ist. Optional soll der Serveradministrator die Mindestgrösse eines Bildes selber festlegen können.

1.5 Extraktion anderer Strukturen (Kannkriterium)

Da bei vielen Dateiformaten häufig auch andere Strukturen (z.B. Tabellen, sonst. Grafiken, Diagramme etc.) anzutreffen sind, wäre es in einem späteren Schritt durchaus sinnvoll, auch diese Informationen zu extrahieren und in den Index aufzunehmen.

1.6 APIs

1.6.1 POI (Sollkriterium)

Für die Extraktion der Inhalte der Office Dokumente (Word, Excel, Powerpoint) soll die Open Source Lösung POI (Poor Obfuscation Implementation) von Apache herbeigezogen werden. Diese JAVA Bibliothek spezialisiert sich vor allem auf die Extraktion, Bearbeitung und Erstellung von älteren Office-Dokumenten (97-2003), welche auf dem proprietären OLE 2 Compound Document Format beruhen.
URL des Projekts: http://poi.apache.org/

1.6.2 PDFBox (Sollkriterium)

PDFBox ist eine Open Source Java Bibliothek für den Umgang mit PDF Dokumenten. Dieses Projekt erlaubt die Erstellung, Manipulation und, was für uns wichtig ist, die Extraktion von Inhalten einer PDF Datei.
URL des Projekts: http://www.pdfbox.org/

1.6.3 Eventuelle Erweiterung unterstützter Dateiformate (Kannkriterium)

Für eine spätere Erweiterung der unterstützten Dateiformate (z.B. mit Microsoft Visio), soll ein Interface geschaffen werden, welches eine möglichst einfache Erweiterung gestattet (mit einer nur minimalen Code Veränderung in anderen Dateien). Es ist vorgesehen, dass der Programmierer eine neue Java Klasse schreibt, welche von einer bestimmten Klasse erbt, sodass er nur noch die abstrakten Methoden implementieren

muss. Welche APIs er seinerseits für die Extraktion benutzt, soll absolut keine Rolle spielen und ihm selber überlassen sein.

2 Automatisierung der Extraktion

Da die Extraktion der Inhalte und die anschliessende Indexierung streng aufeinander angewiesen sind, sollen sie zwar 2 getrennte Arbeitsschritte darstellen, für den Enduser jedoch als einen einzigen zu erkennen sein. Es macht wenig Sinn nur Inhalte einer Datei zu extrahieren, ohne dass diese gleich in den Index aufgenommen werden.

Diese Automatisierung soll über ein Webinterface mit entsprechenden Funktionen erreicht werden.

2.1 Bereitstellung von relevanten Dateien

Der User soll folgende Möglichkeiten haben, eine relevante Datei bereitzustellen:

1. Hochladen einer einzelnen Datei (**Sollkriterium**)
2. Hochladen einer Zip-Datei mit Verzeichnisstruktur (**Sollkriterium**
3. Angeben des Pfades einer öffentlich erreichbaren Datei im WWW, die jeweils auf den Server heruntergeladen wird (**Sollkriterium**)
4. Angeben des Pfades eines öffentlichen Verzeichnisses (**Kannkriterium**)
5. Angeben des Pfades eines lokalen Verzeichnisses (**Kannkriterium**)

2.1.1 Hochladen einer einzelnen Datei (Sollkriterium)

Der User soll über die HTML-Komponente „input file“ eine Datei hochladen können. Hierbei wird gleichzeitig überprüft ob, die Endung der Datei mit einem der unterstützten Dateiformate übereinstimmt. Die Datei wird in einem öffentlich erreichbaren Verzeichnis des Servers abgespeichert, sodass es dem Enduser später ermöglicht wird, diese Datei für den eigenen Gebrauch vom Server herunterzuladen.

2.1.2 Hochladen einer Zip-Datei (Sollkriterium)

Da das Hochladen eines Ordners (oder mehrerer Dateien gleichzeitig) über HTML unmöglich ist, wäre die Unterstützung eines hochladbaren Zip-Files eine gute Lösung. Im Prinzip ist es dasselbe wie das Hochladen einer einzelnen Datei, ausser dass sich in diesem Zip-File eine gesamte Verzeichnisstruktur befinden kann. Die Verzeichnisse sollen rekursiv abgearbeitet werden und die jeweiligen relevanten Dateien im öffentlichen Verzeichnis der Servers abgespeichert werden. Nicht relevante Dateien (nicht unterstützte Dateiformate) sollen ignoriert werden.

2.1.3 Angabe des Pfades einer öffentlich erreichbaren Datei im WWW (Sollkriterium)

Besonders praktisch für den User ist es, eine öffentliche Datei direkt als Pfad anzugeben. Dies erspart dem User das Herunterladen der Datei auf seinen Rechner und das anschliessende wieder Hochladen auf den Server. Hierbei wird die Datei auf unseren Server heruntergeladen, extrahiert, indexiert und anschliessend wieder gelöscht.
Ein nicht zu vernachlässigender Pluspunkt ist, dass die Datei nach dem Herunterladen auf

den Server und der anschliessenden Indexierung wieder gelöscht werden kann, da die Datei bereits auf einem anderen Server erreichbar ist. Dies wirkt sich natürlich extrem platzsparend auf der Festplatte unseres Servers aus.

2.1.4 Angabe des Pfades eines öffentlichen Verzeichnisses (Kannkriterium)

Diese Lösung ist im Prinzip dasselbe wie oben beschrieben, nur das, anstatt des Pfades einer Datei, der Pfad eines Verzeichnisses angegeben werden kann.

2.1.5 Angabe des Pfades eines lokalen Verzeichnisses (Kannkriterium)

Dem User soll es möglich sein, ein Verzeichnis auf dem Server anzugeben, in dem Sich relevante Dateien befinden. Es werden automatisch die Inhalte aller sich in diesem Verzeichnis befindenden relevanten Dateien extrahiert und indexiert.

2.2 Ablauf der Automatisierung

1. **Bereitstellung Datei**
2. **Extraktion der Inhalte (Text, Autor, Bilder)**
3. **Indexierung des Textes und Speicherung von Meta-Informationen**
4. **Indexierung der Bilder**

Nachdem sich der User für eine der Möglichkeiten entschieden hat, eine oder mehrere Dateien bereitzustellen, werden automatisch die benötigten Inhalte der Datei extrahiert und dem Index beigefügt.

2.3 Webinterface

Die automatisierte Extraktion und Indizierung der Daten eines Dokuments soll über ein Webinterface geschehen. Dies bietet den Vorteil einer clientseitigen Plattform- und Standortunabhängigkeit.

2.3.1 Technologie

Als serverseitige Technologie soll JSF (JavaServer Faces) Version 1.2 eingesetzt werden. Diese bietet vor allem den Vorteil einer strikten Implementierung des MVC Prinzips (Model-View-Controller Prinzip), welches eine klare Trennung von Geschäftslogik und Darstellung zulässt.
Voraussichtlich wird die Referenzimplementierung JSFs von Sun verwendet. Dafür soll eigens der Glassfish Server (v2ur2) von Sun installiert werden, welcher normalerweise einen reibungslosen und stabilen Ablauf gewährleisten sollte. Mehr Informationen über Glassfish gibt es unter https://glassfish.dev.java.net/.

2.3.2 Ajax Features

Um den User während des Extraktions- und Indexierungsprozesses über die gängigen Prozesse zu informieren, soll auch eine minimale Ajax-Funktionalität in das Userinterface eingebaut werden.

2.3.2.1 Sollkriterium

Es soll dem User minimale Rückmeldung gegeben werden, welcher Arbeitsschritt gerade in Gange ist. Da AJAX ziemlich aufwändig ist, soll hier nur das nötigste vorhanden sein, sodass der User weiss, dass alles nach Plan läuft.

2.3.2.2 Kannkriterium

Es soll dem User detaillierte Rückmeldung über die ausgeführten Arbeitsschritte geben, wobei für ihn auch der prozentuale Fortschritt zu sehen sein soll.

3 Indexierung/Persistierung von Informationen eines Dokuments

Die Indexierung/Persistierung der Daten eines Dokuments soll in 2 Schritten erfolgen:

1. **Indexierung/Persistierung mit Lucene (für Text und Meta-Informationen)**
2. **Indexierung aller relevanten (kompatiblen) Bilder mit GIFT**

Für die Indexierung/Persistierung wird natürlich vorausgesetzt, dass die Informationen (Text, Meta-Daten und Bilder) bereits extrahiert wurden und bereitstehen.

3.1 Indexierung und Persistierung mit Lucene (Sollkriterium)

Für die Indexierung des Textes existiert bereits ein vollumfängliches und relativ leicht anzuwendendes Java API namens Lucene. Wie viele Open Source Komponenten gehört auch dieses API der Apache Community an.

Zum einen soll natürlich der gesamte Text indexiert werden, andererseits müssen auch andere Meta-Informationen wie Adresse des Files usw. zur Verfügung stehen (ähnlich wie in einer Datenbank). Glücklicherweise ist es hierfür nicht nötig, extra eine Datenbank anzulegen, da Lucene von Haus aus bereits mit solch einer Funktion ausgestattet ist. Im Konkreten heisst das: Es wird für jedes zu indexierende Dokument die Lucene-Klasse „Document“ instanziiert. Danach werden der Instanz laufend Informationen hinzugefügt, welche (wie in einer Datenbank) persistent bleiben. Ob die Informationen nun gespeichert(persistiert) oder indexiert werden (oder gar beides), bleibt dem Programmierer überlassen.

3.1.1 Relevante Informationen für Lucene Index

Art der Information	Indexiert	Gespeichert	Sollkriterium
Text	x		x
Autor	x	x	x
Bildernamen	x	x	x
Dateiname	x	x	x
Adresse des Files		x	x
Abstract (erste 100 Zeichen)		x	x

3.2 Indexierung mit GIFT (Sollkriterium)

GIFT (GNU Image Finder Tool) ist für die Indexierung der Bilder zuständig. Jedes web-kompatible Bild des Dokuments soll mit GIFT indexiert werden. Später soll nach diesen Bildern gesucht werden können aufgrund ihres Inhalts.

Da nur sehr wenige Informationen über GIFT verfügbar sind und ich selber noch praktisch nichts über dieses Tool weiss, wird dieser Teil zu einem späteren Zeitpunkt erweitert.

4 Webinterface für die Suche

Unmittelbar nach der erfolgreichen Indexierung eines Dokuments, ist es möglich , diesen Index zu durchsuchen. Dies soll wiederum über ein Webinterface implementiert in JSF (siehe Punkt 2.3.1) geschehen.

Nach folgenden Inhalten soll gesucht werden können:

1. **Text (Sollkriterium)**
2. **Bilder (Sollkriterium)**
3. Autor (Kannkriterium)

4.1 Suche nach Text

4.1.1 Suchmaske (Sollkriterium)

Der User kann einen Suchbegriff in eine Textbox eingeben. Nach betätigen des Search Buttons wird der Lucene Index durchsucht.
Für erweiterte Suche können spezielle Begriffe aus dem Lucene-Vokabular verwendet werden.

4.1.2 Ergebnisse

Als Antwort auf die Suche erhält der User alle relevanten Dateien mit Zusatzinformationen als Ergebnis.

4.1.3 Sortierung der Ergebnisse

Die Ergebnisse sollen nach Relevanz geordnet sein. Das heisst das relevanteste Suchergebnis soll zuoberst in der Liste stehen.

4.1.4 Zusammensetzung eines Ergebnisses

Ein Ergebnis setzt sich aus folgenden Komponenten zusammen:

1. Dateiname (**Sollkriterium**)
2. Abstract (**Sollkriterium**)
3. Autor (**Sollkriterium**)
4. Enthaltene Bilder (**Sollkriterium**)
5. Downloadlink (**Sollkriterium**)
6. Ähnliche Ergebnisse (**Kannkriterium**)

4.1.4.1 Dateiname (Sollkriterium)

Der Name der Datei zum Zeitpunkt des Hoch- bzw. des Herunterladens.

4.1.4.2 Abstract (Sollkriterium)

Die ersten 100 Zeichen der Datei. Dies soll einen kurzen Überblick geben, welches Thema die Datei behandelt. Es kann nicht sichergestellt werden, dass die ersten 100 Zeichen in einem Text aussagekräftig sind, daher mit Vorsicht zu geniessen.

4.1.4.3 Autor (Sollkriterium)

Sofern während des Extraktionsprozesses der Name des Autors extrahiert werden konnte, wird er beim Suchergebnis aufgelistet.

4.1.4.4 Enthaltene Bilder (Sollkriterium)

Durch einen Klick auf „Enthaltene Bilder“ wird der User auf eine neue Seite umgeleitet, worauf er die Thumbnails (Miniatur-Bilder) der enthaltenen Bilder sieht. Durch einen Klick auf das Thumbnail wird auf einer neuen Seite das Bild in voller Grösse angezeigt.

4.1.4.5 Downloadlink (Sollkriterium)

Entspricht ein aufgelistetes Ergebnis den Wünschen des Users, so hat er die Möglichkeit, die entsprechende Datei herunterzuladen.

4.1.4.6 Ähnliche Ergebnisse (Kannkriterium)

Durch anklicken des Links „Ähnliche Ergebnisse“ (similar results) soll es möglich sein, durch nur einen Klick ähnliche Dokumente (wie das eben angewählte) zu finden.

4.2 Suche nach Bildern

Dem User steht nicht nur eine Suche nach Text, sondern auf eine Suche nach Bildern zur Verfügung. Hierbei wird nicht nach Meta-Informationen des Bildes gesucht, sonder direkt nach dem Bildinhalt.

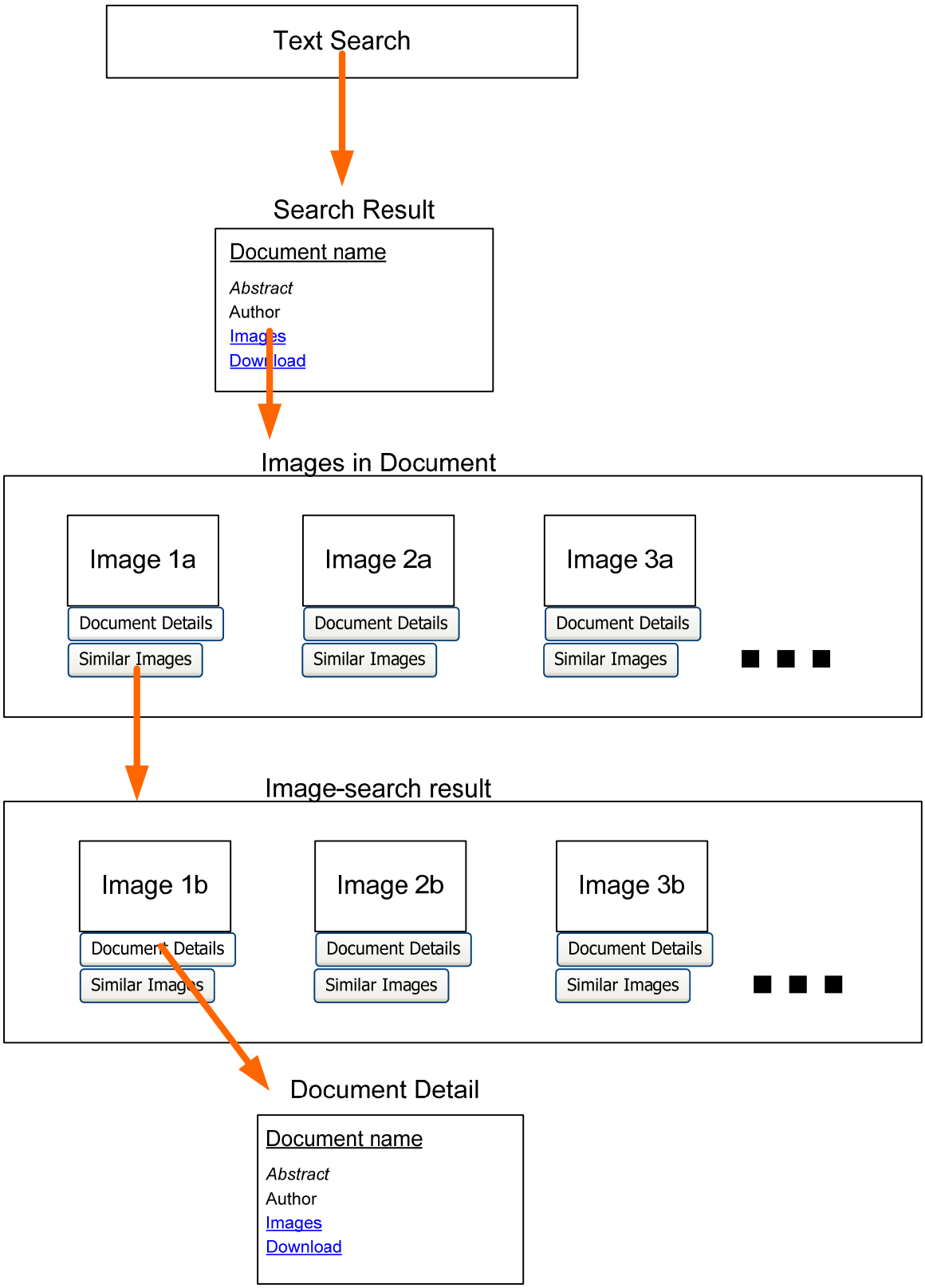
Der User hat dabei folgende Möglichkeiten:

1. **Anzeigen ähnlicher Bilder (Sollkriterium)**
2. **Hochladen eines Bildes (Kannkriterium)**

4.2.1 Anzeigen ähnlicher Bilder (Sollkriterium)

Bei der Suche nach Text und der anschliessenden Anzeige der Ergebnisse besteht die Möglichkeit der Anzeige aller im Dokument enthaltenen Bilder in Thumbnail-Form. In der Nähe eines jeden Thumbnails soll ein Button oder Link platziert werden, welcher die Suche ähnlicher Bilder veranlasst. Diese ähnlichen Bilder werden daraufhin angezeigt und der User kann die Suche wiederum einschränken indem er auf den Button neben dem Bild klickt.
Zusätzlich zum Finden ähnlicher Bilder soll der User in der Lage sein, aufgrund des Bildes zum Dokument zu gelangen (welches das Bild enthält). Hierfür ist wiederum ein Button neben dem Bild vorgesehen, welcher beim Anklicken, die Informationen des Dokuments anzeigt. Um das zu erreichen wird der Lucene Index nach dem Dateinamen des Dokuments abgefragt (Aufgrund des Bildnamens soll auf den Dateinamen des Dokuments geschlossen werden können). Dadurch wird das Ergebnis angezeigt. Dieses Ergebnis ist exakt dasselbe wie bei der Suche nach Text (Punkt 4.1.4), mit der Ausnahme

dass nur ein Ergebnis angezeigt wird. Somit besteht Natürlich automatisch die Möglichkeit, dass der User das Dokument (indem das Bild enthalten ist) herunterladen kann.
Die untenstehende Grafik soll den Suchprozess verdeutlichen.


Text Search
Search Result
Document name
Abstract
Author
Images
Download
Images in Document
Image 1a
Image 2a
Image 3a
Document Details
Similar Images
Image-search result
Image 1b
Image 2b
Image 3b
Document Detail
Document name
Abstract
Author
Images
Download

4.2.2 Hochladen eines Bildes (Kannkriterium)

Die Bildersuche soll dem User ermöglichen, ein Bild hochzuladen. Anhand dieses Bildes sollen nun ähnliche Bilder vorgeschlagen werden. Klickt der User wiederum auf ein vorgeschlagenes Bild, so werden , von diesem ausgehend, wiederum ähnliche Bilder vorgeschlagen (siehe auch obige Grafik). Dieser Prozess wird solange wiederholt, bis der User das gewünschte Bild gefunden hat.

14.4 Wochen-Stunden

Woche	Ausgeführte Arbeiten	Anzahl Stunden pro Woche
Woche 1	• Ausarbeitung des Pflichtenhefts • Einarbeitung in die Materie • Installation VmWare-Image und sonstiger Tools • Beginn mit der Entwicklung der Applikation	45
Woche 2	• Erarbeitung und Implementierung des Konzeptes für die Bereitstellung, Extraktion und Indizierung • Teil-Implementierung der Indizierung eines hochgeladenen Files • Lektüre von Lucene in Action • Treffen mit dem Betreuer	45
Woche 3	• Fertigstellung der Implementierung für die Indizierung eines hochgeladenen Files • Implementierung der Indizierung eines öffentlich erreichbaren Files • Treffen mit dem Betreuer • Implementierung der Indizierung eines Server-Verzeichnisses	45
Woche 4	• Einarbeitung in die textuelle Suche mit Lucene • Implementierung der textuellen Suche • Implementierung der Anzeige aller in einem Dokument enthaltenen Bilder • Treffer mit dem Betreuer • Auseinandersetzung mit GIFT (SnakeCharmer, MRML usw.) • Teil-Implementierung der GIFT-Suche	45
Woche 5	• Teil-Implementierung für die GIFT-Suche • Layout-Design der Applikation • Dokumentation	52
Woche 6	• Fertigstellung der Implementierung für die GIFT-Suche	49

	• Anzeige der Details der einem Bild zugehörigen Dokument • Implementierung der Suche ähnlicher Resultate (textuelle Suche) • Testen der Applikation • Treffen mit dem Betreuer • Dokumentation	
Woche 7	• Treffen mit dem Betreuer • Übergabe des VMWare-Images an den Informatik-Dienst • Testen der Applikation • Letzte Verbesserungen an der Applikation • Dokumentation	55
Woche 8	• Treffen mit dem Betreuer • Dokumentation	53
Stunden Total		**389**

15 Index

Q

R

S

T

U

V

W

Z